UNIVERSIDAD NACIONAL DEL ALTIPLANO

FACULTAD DE INGENIERÍA MECÁNICA ELÉCTRICA,

ELECTRÓNICA Y SISTEMAS

ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS

"ANÁLISIS Y OPTIMIZACIÓN DEL ALGORITMO DE ENCRIPTACIÓN RIJNDAEL EN EL QUE SE BASA EL ESTÁNDAR DE ENCRIPTACIÓN AVANZADA AES (ADVANCED ENCRYPTION STANDARD)"

TESIS

PRESENTADO POR:

JUAN CARLOS DÁVILA TORRES

PARA OPTAR EL TÍTULO PROFESIONAL DE:

INGENIERO DE SISTEMAS

Puno – Perú

2017

Universidad Nacional del Altiplano

FACULTAD DE INGENIERÍA MECÁNICA ELÉCTRICA, ELECTRÓNICA Y SISTEMAS

ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS

ANÁLISIS Y OPTIMIZACIÓN DEL ALGORITMO DE ENCRIPTACIÓN RIJNDAEL EN EL QUE SE BASA EL ESTÁNDAR DE ENCRIPTACIÓN AVANZADA AES (ADVANCED ENCRYPTION STANDARD)

TESIS PRESENTADA POR:

JUAN CARLOS DÁVILA TORRES

PARA OPTAR EL TÍTULO PROFESIONAL DE INGENIERO DE SISTEMAS

APROBADA POR EL JURADO REVISOR CONFORMADO POR:

PRESIDENTE: .....
M.Sc. William Eusebio Arcaya Coaquira

PRIMER MIEMBRO: .....
Mg. Ing. Robert Antonio Romero Flores

SEGUNDO MIEMBRO: .....
Ing. Yalmar Temistocles Ponce Atencio

DIRECTOR DE TESIS: .....
Dr. Henry Iván Condori Alejo

Puno – Perú
2017

ÁREA: INFORMÁTICA
TEMA: SEGURIDAD INFORMÁTICA
LÍNEA: CRIPTOGRAFIA

ÍNDICE

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

íNDICE DE ANEXOS

RESUMEN

La presente investigación tiene como objetivo principal, proponer una modificación del algoritmo Rijndael de forma que su estructura sea más consistente y robusta contra los ataques de fuerza bruta, asegurando la aleatoriedad de los datos. Se analizó complejidad computacional del algoritmo original, con la ayuda del análisis temporal de algoritmos; se modificó la función KeyExpansion donde se genera las claves necesarias para las rondas según el tamaño de la clave con la modificación esta también calcula los movimientos para cada ronda, y ShiftRow que es una de las funciones principales para la difusión de caracteres, en la modificación de esta función se realizan los movimientos según los calculados en la función KeyExpansion.

Después de realizada la modificación se analizó asintóticamente la modificación planteada para finalmente comparar los resultados con los obtenidos en el análisis asintótico del algoritmo original.

PALABRAS CLAVE: Criptografía, AES, algoritmo de compresión de Huffman, algoritmos de encriptación, análisis del algoritmos, criptografía simétrica

ABSTRACT

The main objective of this research is to propose a modification of the Rijndael algorithm so that its structure is more consistent and robust against brute force attacks, ensuring the randomness of the data computational, complexity of the original algorithm was analyzed with the help of temporal analysis algorithms.

The function KeyExpansion was modified where the necessary keys are generated for the rounds according to the size of the key with the modification, this also calculates the movements for each round and ShiftRow which is one of the main functions for the diffusion of characters, in the modification of this function the movements are made according to those calculated in the KeyExpansion function. After the modification was performed, the modification was analyzed asymptotically to finally compare the results with those obtained in the asymptotic analysis of the original algorithm.

KEYWORDS: Cryptography, AES, Huffman compression algorithm, encryption algorithms, analysis of the algorithm, symmetric Cryptography

INTRODUCCIÓN

En 1997, el Instituto Nacional de Normas y Tecnología (NIST) decidió realizar un concurso para escoger un nuevo algoritmo de cifrado capaz de proteger información sensible durante el siglo XXI. Este algoritmo se denominó Advanced Encryption Standard (AES). El algoritmo Rijndael ganó el concurso y en noviembre de 2001 se publicó FIPS 197 donde se asumía oficialmente, actualmente el algoritmo AES es uno de los algoritmos más populares usados en criptografía simétrica.

No podemos prever con certeza los nuevos desarrollos computacionales técnicos o teóricos. Para predicciones computacionales, la “ley de Moore” es asumida: En términos generales, establece que el poder de cómputo se duplica cada 18 meses, mientras que los costos permanecen constantes. Esto tiene muchas implicaciones en la criptografía, ya que después de 10 iteraciones (15 años), podremos hacer 2^{10}=1024 veces más cálculos computacionales de los que hoy podemos por la misma cantidad de dinero.

La presente investigación inicia con un planteamiento de los problemas actuales en la criptografía moderna, que se describirán en el **capítulo 1** así como su impacto en la seguridad informática, en donde nos centraremos específicamente en el algoritmo AES y sus problemas, analizaremos sus funciones y complejidad computacional, para después plantear una modificación de sus funciones para también analizar su complejidad, compararla con el algoritmo original.

En el **capítulo 2** están descritos los aspectos específicos teóricos en los que basamos esta investigación, como otros trabajos encontrados sobre

encriptación, la descripción del algoritmo AES, las herramientas que se usaran para el análisis de este, en base a los planteamientos teóricos se plantea la hipótesis general, especificas así como la operazionalización de variables.

En el **capítulo 3** se describe el diseño metodológico de la investigación, la población y la muestra tomadas para la investigación así como las técnicas e instrumentos utilizados en esta investigación.

En el **capítulo 4** se describirá la implementación del algoritmo, también de su modificación así como los distintos tipos de análisis a los que se sometieron.

Para finalizar en el capítulo 5 tendremos el análisis y la interpretación de los resultados obtenidos en la investigación.

CAPÍTULO I.

PLANTEAMIENTO DEL PROBLEMA DE INVESTIGACIÓN

1.1. DESCRIPCIÓN DEL PROBLEMA

Desde la aparición de las computadoras y más aun con el crecimiento de las redes, la necesidad de intercambiar información de manera segura es mayor y es donde aparecen implementaciones de distintos sistemas de cifrado. Existen dos tipos de algoritmos de cifrado que involucran distintos tipos de algoritmos, estos son los sistemas simétricos y los sistemas asimétricos, ambos con sus ventajas y sus desventajas.

Entre los algoritmos de cifrado simétrico se encuentra el algoritmo AES (Advanced Encryption Standard) o también llamado Rijndael que es uno de los algoritmos más utilizados en la actualidad, considerado por el gobierno de los Estados Unidos como un algoritmo seguro para protección nacional de información, desarrollado en 1997 por Joan Daemen y Vincent Rijmen.

El 3 de diciembre del 2012 en la conferencia de Passwords^12 en Oslo, que se realizó en Noruega fue develada una de las técnicas más avanzadas para descifrar contraseñas bajo los algoritmos Hash LM, NTLM, MD5, SHA-1, bcrypt (05) y sha512crypt. El procedimiento que antes tomaba semanas con contraseñas complejas, ahora toma pocas horas o minutos dependiendo del algoritmo según el portal “welivesecurity.com Noticias, opiniones y análisis de la comunidad de seguridad de ESET”.

En el 2014 la compañía de seguridad Trustwave demostró que una placa Radeon 7970 de AMD es capaz de calcular a cada segundo 17.300 millones de

operaciones de hashing para contraseñas basadas en el protocolo de NT LAN Manager. Y cuesta 350 dólares. Los investigadores de Trustwave usaron dos equipos para sus experimentos, uno con 2 Radeon 7970 y otro con 4. El costo total de ambas máquinas fue de 4500 dólares, incluyendo el resto del hardware (microprocesador, memoria RAM). Es decir, ya no hace falta ni una supercomputadora ni es necesario esperar que lleguen los chips cuánticos. Con menos de 5000 dólares, Trustwave logró quebrar en los primeros minutos el 54% de las 626.718 contraseñas que tenían por objeto descubrir. Luego, en el curso de escasos 31 días, consiguieron descubrir casi el 92% de las claves (576,533) cita el periódico “la nación” de argentina.

En agosto del año 2014 un equipo de investigadores ha encontrado la primera vulnerabilidad en el estándar de cifrado AES reduciendo la longitud efectiva de la clave en 2 bits. Esto implica que las longitudes habituales de 128, 192 y 256 bits se han visto reducidas a 126, 190 y 254 bits respectivamente. En el sentido estrictamente académico, en algoritmo está "roto" puesto que se ha reducido (aunque sea en 2 bits) el espacio de claves necesario para calcular la clave por fuerza bruta.

1.1.1. PROBLEMA GENERAL

Mejorar la efectividad de cifrado y descifrado con la modificación al algoritmo Rijndael en el que se basa el estándar de encriptación avanzada AES (Advanced Encryption Standard)

1.1.2. PROBLEMAS ESPECIFICOS

- ¿El análisis de la complejidad de algoritmos, análisis de entropía de salidas y el análisis de las posibles salidas de la función ShiftRows nos ayuda a comprender mejor el algoritmo Rijndael y ubicar sus debilidades?
- ¿Las funciones diseñadas, no tienen ningún tipo de perdida de información y aseguran una mayor aleatoriedad de los datos sin que se incremente la complejidad temporal del algoritmo?
- ¿Al implementar las funciones en el algoritmo original no se genera ninguna alteración en el funcionamiento de este en la generación de las cadenas cripticas?
- ¿Al analizar las funciones implementadas y comparándolas con el algoritmo original, se muestra una mayor aleatoriedad y seguridad de los datos, sin que se pierda la eficiencia en tiempo de ejecución?

1.2. JUSTIFICACIÓN DE LA INVESTIGACIÓN

La tecnología de hoy en día es capaz de elevar las tasas de pruebas por fuerza bruta a valores muy altos; no podemos prever con certeza los nuevos desarrollos computacionales técnicos o teóricos. Para predicciones computacionales, la “ley de Moore” es asumida: En términos generales, establece que el poder de cómputo se duplica cada 18 meses, mientras que los costos permanecen constantes. Esto tiene muchas implicaciones en la criptografía, ya que después de 10 iteraciones (15 años), podremos hacer 2^{10}=1024 veces más cálculos computacionales de los que hoy podemos por la

misma cantidad de dinero; nos vemos en la necesidad de mejorar los algoritmos de cifrado actuales ya que son parte fundamental de la seguridad informática.

Además en agosto del año 2014 un equipo de investigadores ha encontrado la primera vulnerabilidad en el estándar de cifrado AES reduciendo la longitud efectiva de la clave en 2 bits. Esto implica que las longitudes habituales de 128, 192 y 256 bits se han visto reducidas a 126, 190 y 254 bits respectivamente. En el sentido estrictamente académico, en algoritmo está "roto" puesto que se ha reducido (aunque sea en 2 bits) el espacio de claves necesario para calcular la clave por fuerza bruta. (Borja & De Los Santos, 2011)

1.3. OBJETIVOS DE LA INVESTIGACIÓN

1.3.1. OBJETIVO GENERAL

Proponer una modificación del algoritmo Rijndael cuya estructura sea más robusta, asegurando una mayor aleatoriedad de los datos sin incrementar la complejidad temporal.

1.3.2. OBJETIVOS ESPECÍFICOS

- Analizar las funciones del algoritmo Rijndael mediante la técnica de análisis de la complejidad temporal, la entropía de sus salidas mediante la comprensión de los archivos encriptados y la cantidad de posibles salidas de la función ShiftRows.
- Diseñar las modificaciones para las funciones KeyExpansion y ShiftRows que aseguren una mayor aleatoriedad de los datos.
- Implementar las nuevas funciones KeyExpansion y ShiftRows al algoritmo Rijndael.

- Analizar la modificación del algoritmo con el análisis de la complejidad temporal de algoritmos y comprensión de los archivos encriptados con el algoritmo de Huffman, para su posterior comparación con el algoritmo original.

CAPÍTULO II.

MARCO TEÓRICO

2.1. ANTECEDENTES DE INVESTIGACIÓN

a) "DESCRIPCIÓN POLINOMIAL DE LOS SISTEMAS DE CIFRADO DES Y AES" GARCÍA MÉNDEZ, PAULO SERGIO UNIVERSIDAD AUTÓNOMA METROPOLITANA – MÉXICO 2011

En este trabajo se ha presentado una descripción polinomial del sistema de cifrado DES. Esta especiación es una contribución original basada en la representación polinomial que Joachim Rosenthal hace sobre el sistema de cifrado AES. En ésta última, se usan polinomios de permutación y una técnica para calcularlos conocida como Fórmula de Interpolación de Lagrange. (GARCÍA MENDEZ, 2011) La técnica anterior fue aplicada en la mayoría de las permutaciones del sistema DES para lograr una forma polinomial de éstas. A continuación, se repasan algunos de los logros más importantes que conforman la representación polinomial del sistema DES.

Un resultado importante fue la representación de las S-cajas. Como se recordará, en estas tablas reside la seguridad del sistema DES, los polinomios calculados tienen cocientes en el campo $GF(2^4)$ y se requirieron 4 para representar cada S-caja.

Finalmente, se ha proporcionado tanto un algoritmo de cifrado como uno de descifrado del sistema DES en términos de operaciones polinomiales dentro de varios campos finitos y algunos anillos de polinomios. Todo esto conforma las representación polinomial del sistema de cifrado DES.

Como trabajos futuros, podemos señalar dos opciones principales. El primero consiste en seguir representando otros sistemas de cifrado conocidos con el objetivo de reconocer patrones o características que puedan servirnos de guía hacia la construcción o diseño de nuevas S-cajas. La segunda opción plantea encontrar condiciones para determinar cuáles y cuántos de todos los mapeos de un campo finito en sí mismo son polinomios de permutación los cuales son todo un tema de investigación muy importante y con muchas aplicaciones, por ejemplo, en la Teoría de Códigos, Criptografía, Turbo códigos, entre otros.

b) "DESARROLLO DE ENCRIPTADO AES EN FPGA" C. LIBERATORI, MÓNICA UNIVERSIDAD NACIONAL DE LA PLATA – ARGENTINA 2006

Al finalizar su investigación Liberatori (C. LIBERATORI, 2006) Concluyo que La implementación de cifradores de bloques enfrenta al diseñador con diversas estrategias. La decisión que influye sobre la elección se relaciona con la característica o parámetro de diseño que se desee optimizar. En el caso de diseño de cifradores en hardware, la elección buscará optimizar el área ocupada por el circuito, la velocidad de procesamiento del mismo o la latencia asociada. Cada elección de diseño impone diferentes restricciones sobre la implementación final, existiendo un compromiso particular entre dos parámetros característicos del diseño final: área y velocidad. En este sentido no puede mejorarse uno sin la consiguiente penalización en términos del otro.

Cualquiera sea la estrategia, todas comparten a una estructura básica en cuanto a la organización de los diversos componentes del circuito. Esa estructura

básica podrá distinguirse por el reconocimiento de unidades comunes a todos los circuitos cifradores: una unidad de cifrado/descifrado siempre presente, una unidad opcional de tratamiento de la clave que podrá reemplazarse por una unidad de memoria para almacenamiento de la clave original y las subclaves derivadas de la misma, una interfaz para comunicación con el mundo exterior e ingreso de los datos, una interfaz para almacenamiento de los datos cifrados y su envío posterior al exterior y una unidad de control que coordina el funcionamiento de las demás unidades mencionadas.

c) "ENCRIPTACIÓN RSA DE ARCHIVOS DE TEXTO" LEON LOMPARTE KATIA REGINA PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERU (2005).

Según LEON LOMPARTE (LEON LOMPARTE, 2005) Los algoritmos de encriptación tienen una base fuertemente matemática y de esto no escapa el RSA, poder manejar estos conceptos dentro de este programa no ha sido precisamente de las tareas más sencillas, sino por el contrario demandó mucho tiempo no solo en entender el funcionamiento del algoritmo sino también en la implementación y pruebas del mismo.

Siendo el algoritmo implementado una técnica de encriptación muy elaborada, empezaremos esta introducción comentando muy brevemente sobre el nacimiento de la criptografía. Los códigos secretos han sido utilizados desde la antigüedad para enviar mensajes seguros, por ejemplo en tiempo de guerra o de tensiones diplomáticas. Hoy día se guarda, con frecuencia, información delicada de naturaleza médica o financiera en los ordenadores, y es importante mantenerla en secreto.

Muchos códigos están basados en teoría de números. Uno muy simple es sustituir cada letra del alfabeto por la siguiente o por tres letras después. Estos códigos son fáciles de romper pues se pueden probar todos los posibles valores hasta obtener un mensaje comprensible, o podemos comparar las letras más frecuentes en el mensaje con las que se saben que son más frecuentes en la lengua original.

En este trabajo se presenta el método desarrollado en 1978 por R. L. Rivest, A. Shamir y L. Adleman y que es conocido como sistema criptográfico RSA por las iniciales de sus autores. Basa su seguridad en la dificultad de factorizar números primos muy grandes aunque como todo sistema de encriptación de clave pública, el RSA puede estar sujeto a ataques con el fin de obtener el mensaje original o descubrir la clave privada.

d) "ASPECTOS DE SEGURIDAD DE BITCOIN Y SU APLICACIÓN EN UNA ALTERNATIVA DE INFRAESTRUCTURA DE LLAVE PÚBLICA" BASURTO BECERRA, ABRAHAM JESÚS, CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS DEL INSTITUTO POLITÉCNICO NACIONAL – MÉXICO 2015

De las conclusiones de BASURTO BECERRA (BASURTO BECERRA, 2015) de este trabajo se puede tener una amplia visión de cómo la criptografía es empleada en el mundo real, desde cuando navegamos en Internet hasta cuando realizamos pagos a través de una criptomoneda. Además de las implicaciones prácticas y problemas que surgen, como lo es la distribución y verificación de autenticidad de las llaves.

Bitcoin no posee un una formalización de su modelo de seguridad, ha habido varias propuestas pero tienen un enfoque limitado. Es importante el investigar en este tema ya que hasta el momento no se puede probar de manera convincente si el modelo de Bitcoin es seguro o no.

También se puede apreciar el gran problema que existe con la actual implementación de la infraestructura de llave publica y lo necesario que es el encontrar alternativas viables, para de esta manera poder asegurar la confidencialidad e integridad en nuestras comunicaciones.

Como conclusión final se puede indicar que la aportación mas importante que ha brindado Bitcoin no es como tal la criptomoneda, sino la idea del blockchain la cual apenas empieza a permear en otros sectores diferentes al de la ciencia, un ejemplo es el banco español Santander el cual ha buscado usos potenciales para esta tecnología y en un reporte [Belinky et al., 2015] indica que tiene el potencial de poder reducir costos de operación en hasta 20 mil millones de dólares al año para los bancos en las operaciones de pagos internacionales, bonos y cumplimento regulatorio.

e) "ADVANCED ENCRYPTION STANDARD (AES) AND IT'S WORKING" SHRIPAL RAWAL INTERNATIONAL RESEARCH JOURNAL OF ENGINEERING AND TECHNOLOGY (IRJET) – MUMBAI, MAHARASHTRA, INDIA 2016

Del abstract, el cifrado es una pieza interesante de la tecnología que funciona por cifrado de datos por lo que es ilegible por partes no deseadas. El Advanced Encryption Standard (AES), también conocido como Rijndael (su

nombre original), es una especificación para el cifrado de datos electrónicos. El cifrado es un proceso de codificación de mensajes o información vital de tal manera que sólo las partes canónicas pueden leerlo. El cifrado no impide por sí mismo la interceptación, pero niega la información al interceptor. Cifrado en palabras simples significa generar un texto cifrado que sólo puede ser leído por el que tiene clave de descifrado. Una de estas técnicas de cifrado utilizada para proteger los datos en línea de cualquier amenaza maliciosa es Advanced Encryption Standard (AES). En general, el cifrado utiliza esquemas de cifrado de clave simétrica o esquemas de cifrado de clave pública. (Rawal, 2016)

En las conclusiones podemos encontrar, en la criptografía actual, AES es ampliamente adoptado y soportado en hardware y software. Hasta la fecha, no se han descubierto ataques criptoanalíticos contra AES. Además, AES tiene flexibilidad incorporada de longitud de clave, lo que permite un grado de "prueba de futuro" contra el progreso en la capacidad de realizar exhaustivas búsquedas de claves.

Sin embargo, al igual que para DES, la seguridad AES está asegurada sólo si se implementa correctamente y se emplea una buena gestión de claves.

2.2. SUSTENTO TEÓRICO.

Aquí se describe el contexto general teórico existente sobre el algoritmo así como sus principales conceptos que serán abordados a lo largo del desarrollo de la investigación

2.2.1. CRIPTOGRAFÍA

La palabra *criptografía* viene del griego *cripto* (que significa «ocultar») y *graphos* (que significa «escribir») (ROA BUENDÍA, 2013). Se podría traducir por: **cómo escribir mensajes ocultos.** En la antigüedad se utilizaba sobre todo durante las guerras, para comunicar estrategias, de manera que, aunque el mensajero fuera interceptado por el enemigo, el contenido del mensaje estaba a salvo.

La criptografía consiste en tomar el documento original y aplicarle un **algoritmo** cuyo resultado es un nuevo documento. Ese documento está cifrado: no se puede entender nada al leerlo directamente. Podemos, tranquilamente, hacerlo llegar hasta el destinatario, que sabrá aplicar el algoritmo para recuperar el documento original.

2.2.2. CRIPTOGRAFÍA SIMÉTRICA

Los **algoritmos de criptografía simétrica** utilizan la **misma clave** para los dos procesos: cifrar y descifrar. Son sencillos de utilizar y, en general, resultan bastante **eficientes** (tardan poco tiempo en cifrar o descifrar). (ROA BUENDÍA, 2013) Por este motivo, todos los algoritmos, desde la antigüedad hasta los años setenta, eran simétricos. Los más utilizados actualmente son DES, 3DES, AES, Blowfish e IDEA.

El funcionamiento es simple: en la figura el emisor quiere hacer llegar un documento al receptor. Toma ese documento y le aplica el algoritmo simétrico, usando la clave única, que también conoce el receptor. El resultado es un documento cifrado que ya podemos enviar tranquilamente.

Cuando el receptor recibe este documento cifrado, le aplica el mismo algoritmo con la misma clave, pero ahora en función de descifrar. Si el documento cifrado no ha sido alterado en el camino y la clave es la misma, el resultado será el documento original.

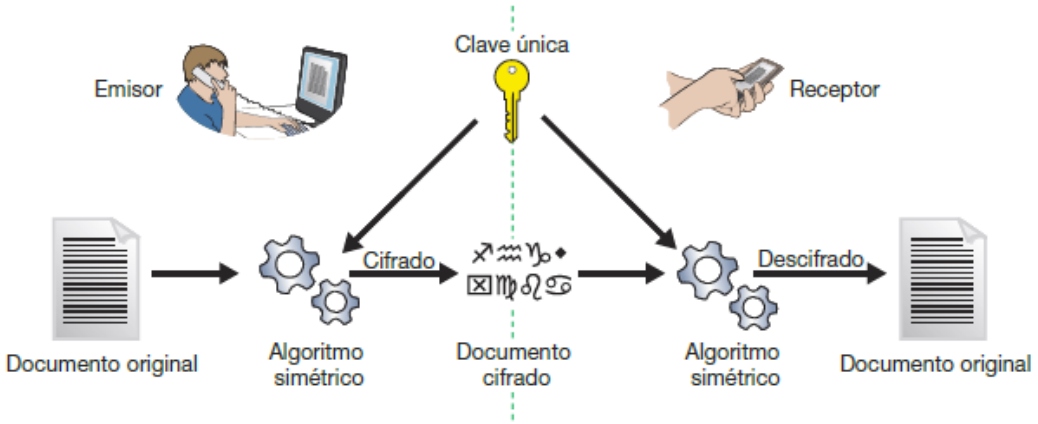


Figura 1: Criptografía simétrica.

Fuente: (ROA BUENDÍA, 2013)

2.2.2.1. Problemas de la criptografía simetrica

El **problema principal** de la criptografía simétrica es la **circulación de las claves:** cómo conseguimos que el emisor y el receptor tengan la clave buena. No podemos utilizar el mismo canal inseguro por el que enviaremos el mensaje (la inseguridad nos ha llevado a cifrar). Hay que utilizar un **segundo canal de comunicación,** que también habría que proteger, y así sucesivamente. Por ejemplo, en el correo de bienvenida a una empresa puede aparecer la contraseña de la wifi de la oficina; cuando se cambie, se envía otro correo, etc.

El **segundo problema** es **la gestión de las claves almacenadas**. (ROA BUENDÍA, 2013) Si en una empresa hay diez trabajadores y todos tienen conversaciones privadas con todos, cada uno necesita establecer nueve claves

distintas y encontrar nueve canales seguros para actualizarlas cada vez (en total 81 claves y 81 canales). Si aparece un trabajador nuevo, ahora son 100 claves y 100 canales. Y las empresas pueden tener muchos trabajadores: 500, 5 000, 50 000... ¿Cada vez que cambie mi clave tengo que avisar a 49 999 compañeros? Es poco manejable

2.2.3. CRIPTOGRAFÍA ASIMÉTRICA

En los años setenta, los criptógrafos Diffie y Hellman publicaron sus investigaciones sobre **criptografía asimétrica.** Su algoritmo de cifrado utiliza **dos claves matemáticamente relacionadas** de manera que **lo que cifras con una solo lo puedes descifrar con la otra.** Comparado con la clave simétrica, ahora el emisor no necesita conocer y proteger una clave propia; es el receptor quien tiene el par de claves. Elige una de ellas (llamada **clave pública**) para comunicarla al emisor por si quiere enviarle algo cifrado. Pero ya no hace falta buscar canales protegidos para enviarla porque, aunque un tercer individuo la conozca, todo lo que se cifre con esa clave solo se podrá descifrar con la otra clave de la pareja (la **clave privada**), que nunca es comunicada. (ROA BUENDÍA, 2013) Y matemáticamente es imposible deducir la clave privada conociendo solo la clave pública.

Como se ilustra en la figura, cuando el emisor quiere hacer llegar un mensaje confidencial al receptor, primero consigue la clave pública del receptor. Con esa clave y el documento original, aplica el algoritmo asimétrico. El resultado es un documento cifrado que puede enviar al receptor por cualquier canal.

Cuando el mensaje cifrado llega al receptor, él recupera el documento original aplicando el algoritmo asimétrico con su clave privada.

Si el receptor quiere enviar al emisor una respuesta cifrada, debería conseguir la clave pública del emisor y seguir el mismo procedimiento.

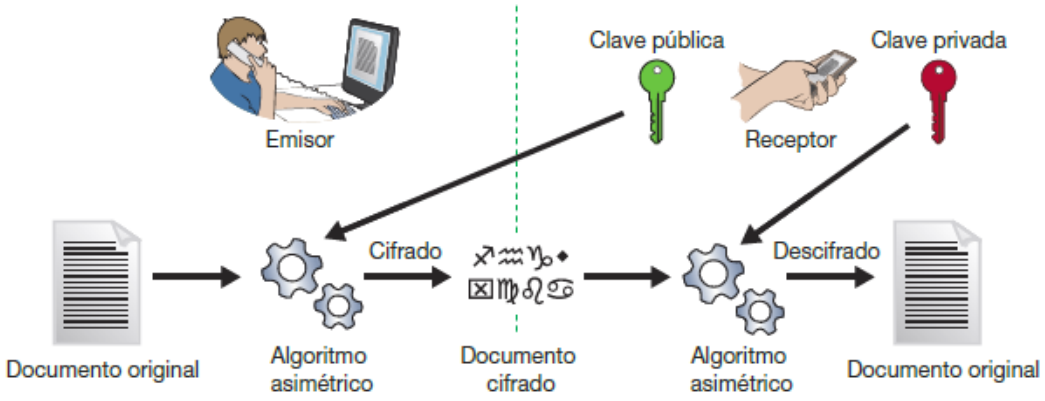


Figura 2: Criptografía asimétrica

Fuente: (ROA BUENDÍA, 2013)

La criptografía asimétrica resuelve los dos problemas de la clave simétrica (ROA BUENDÍA, 2013):

- No necesitamos canales seguros para comunicar la clave que utilizaremos en el cifrado. Podemos adjuntarla en nuestros correos, añadirla al perfil de nuestras redes sociales, «postearla» en un blog, incluso repartirla en octavillas por la calle.
- No hay desbordamiento en el tratamiento de claves y canales. Si somos nueve empleados, solo necesitamos nueve claves y un solo canal: la intranet de la empresa, (ROA BUENDÍA, 2013) un correo destinado a toda la empresa, etc. Y si aparece un empleado nuevo, serán diez claves y el mismo canal.

2.2.3.1. Problemas de los algoritmos asimetricos

Sin embargo, los algoritmos asimétricos tienen sus propios problemas:

- Son **poco eficientes:** tardan bastante en aplicar las claves para generar los documentos cifrados, sobre todo porque las claves deben ser largas para asegurar la independencia matemática entre ellas.
- Utilizar las claves privadas repetidamente es arriesgado porque algunos **ataques criptográficos** se basan en analizar paquetes cifrados. Estos paquetes serían capturados en la red o directamente el atacante podría elaborar un software malicioso que generase paquetes de tamaño y contenido elegidos cuidadosamente y conseguir enviarlos a nuestro servidor para que los devolviera cifrados con su clave privada.
- Hay que **proteger la clave privada.** No basta con dejarla en un fichero de una carpeta del disco duro en la cuenta de nuestro usuario; cualquier otro usuario con permisos de administrador podría llegar hasta él. Por este motivo, las claves privadas se guardan todas juntas en un fichero llamado **keyring** (archivo de llaves, llavero), y este fichero está protegido mediante **cifrado simétrico.** Es decir, para poder usar la clave privada, hay que introducir una clave que descifra el llavero y permite leerla.

Necesitamos una segunda medida de protección de la clave privada: la **copia de seguridad del llavero.** Si el disco duro se estropea, perderemos el fichero que contiene la clave privada y no podremos volver a utilizarla. Por tanto,

debemos incluirlo en la política de backup de la empresa, y confiamos en que, aunque alguien más tenga acceso al backup (cintas, discos, etc.), la clave simétrica todavía protege el llavero.

- Hay que **transportar la clave privada.** En cifrado simétrico, si hemos enviado el fichero cifrado a otra máquina y queremos descifrarlo, basta con recordar la clave e introducirla. Pero en la clave privada esto es imposible (son cientos de símbolos sin sentido). Debemos transportar el llavero, con el riesgo que supone (si lo perdemos, podrían intentar un ataque de fuerza bruta contra el cifrado simétrico).

La solución más común a los problemas de proteger y transportar la clave privada es la **tarjeta inteligente**. Es una tarjeta de plástico que contiene un **chip** electrónico.

Hay dos tipos:

- Tarjeta de **memoria.** Es equivalente a una memoria Flash y se limita a almacenar el llavero. Cuando se introduce en el lector, el ordenador hace una copia temporal del llavero y trabaja con él introduciendo la clave simétrica, etc.
- Tarjeta **procesadora.** La tarjeta de memoria es peligrosa porque hemos expuesto nuestro llavero. En cambio, en las tarjetas procesadoras las claves también están almacenadas, pero nunca salen de la tarjeta. Cualquier cifrado que necesite nuestra clave privada es **realizado por el propio chip** porque incluye una CPU,

memoria RAM, etc. Por supuesto, sigue siendo necesario introducir la clave simétrica que abre el llavero.

El ejemplo más sencillo es la tarjeta SIM de los teléfonos móviles: para poder usarla necesitamos introducir el número PIN. Aunque la usemos en otro teléfono, el PIN es el mismo porque está asociado a la tarjeta. En la sección final de esta unidad veremos otro ejemplo de tarjeta inteligente: el DNI electrónico.

Las tarjetas inteligentes también se pueden clasificar por su tipo de interfaz:

- Tarjeta de **contacto.** El lector necesita tocar los contactos metálicos del chip para interactuar con él. Son las más utilizadas, sobre todo en entornos de alta seguridad, como el sector bancario, Administración electrónica, etc.
- Tarjeta **sin contacto.** El lector utiliza tecnologías inalámbricas para interactuar con el chip. Se utilizan en situaciones donde se necesitan transacciones rápidas, como el acceso al transporte público.

El cifrado asimétrico no se puede utilizar para cifrar todos los paquetes intercambiados en una red local porque el bajo rendimiento del algoritmo ralentizaría el tráfico. En su lugar se adopta un **esquema híbrido:**

- Criptografía **asimétrica solo para el inicio de la sesión,** cuando hay que generar un canal seguro donde acordar la clave simétrica aleatoria que se utilizará en esa conversación.
- Criptografía **simétrica durante la transmisión,** utilizando la clave simétrica acordada durante el inicio de sesión. Generalmente se suele cambiar la clave simétrica cada cierto tiempo (minutos) para dificultar más el espionaje de la conversación.

Es decir, cuando A quiere establecer una conversación con B, en A se genera en ese instante una nueva clave simétrica (CS). Para enviársela a B de modo seguro, A la cifra utilizando un algoritmo asimétrico con la clave pública de B. Cuando B recibe la CS cifrada, la descifra con su clave privada y desde ese momento pueden seguir el diálogo cifrando con el algoritmo simétrico acordado y la CS recibida. En la figura vemos un ejemplo con el protocolo SSH.

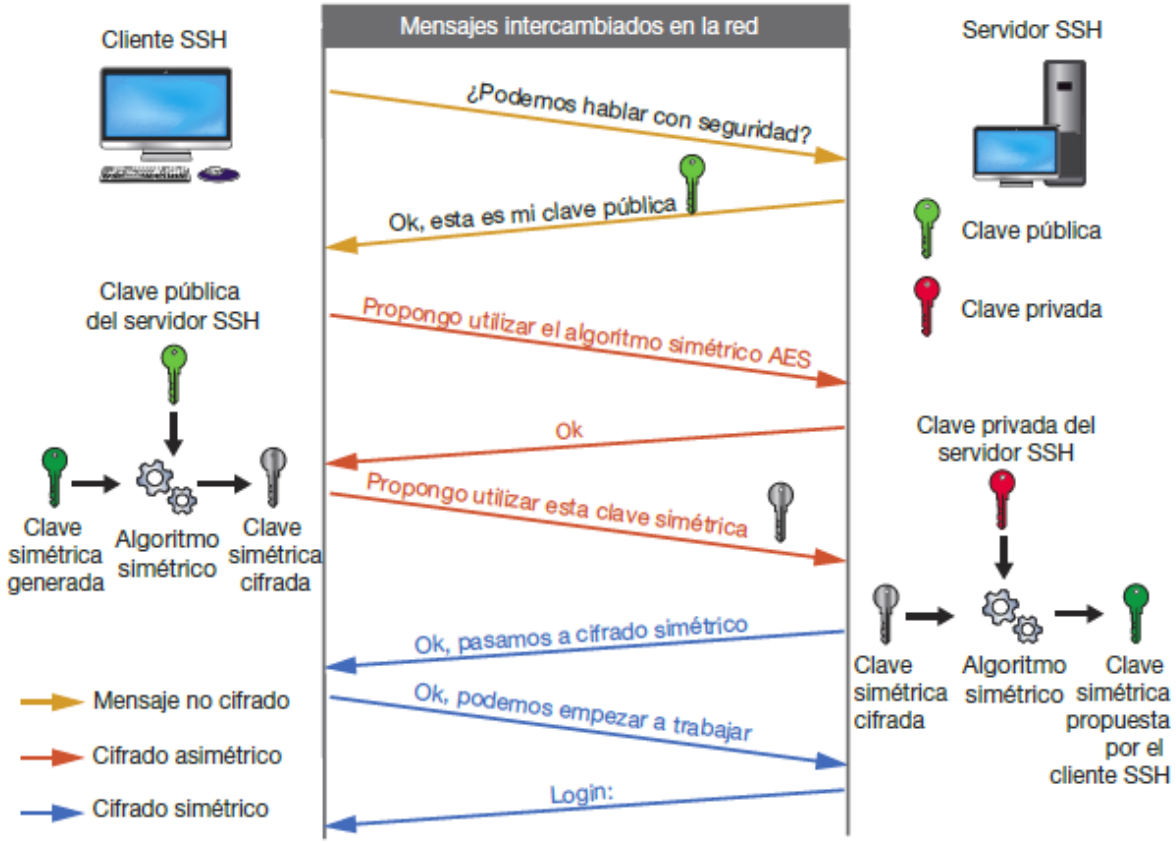


Figura 3: Esquema híbrido de cifrado en SSH.

Fuente: (ROA BUENDÍA, 2013)

2.2.4. CIFRAR Y FIRMAR

La primera utilidad de la criptografía es ocultar el mensaje para aquellos que no son destinatarios del mismo. (ROA BUENDÍA, 2013) Es decir, garantizar la **confidencialidad de la comunicación** cifrando el documento original.

La segunda utilidad es conseguir determinar la **autenticidad del emisor.** ¿Cómo podía estar seguro el general romano de que ese mensaje con las nuevas órdenes venía de otro general romano, y no de algún enemigo? Si el enemigo conocía el algoritmo de cifrado y la clave actual, podía intentar engañarle mediante un mensaje falso pero correctamente cifrado.

En criptografía asimétrica, el **mecanismo de firma** garantiza que el emisor es quien dice ser. Supongamos que vamos a enviar un documento y queremos que el receptor confíe en que somos nosotros. Para conseguirlo, el emisor aplica al documento una función resumen (función **hash**). El resultado de esta función es una lista de caracteres, el **resumen,** que la función garantiza que solo se pueden haber obtenido con el documento original (el algoritmo de la función hash no necesita una clave extra como los algoritmos de cifrado). Ahora **el emisor cifra ese resumen con su clave privada** y lo envía al destino, junto con el documento original.

En el destino se hacen dos operaciones:

- Aplicar la misma función hash al documento para obtener su resumen.
- Descifrar el resumen recibido, utilizando la clave pública del emisor.

Si ambos resúmenes coinciden, el destino puede estar seguro de que el emisor del documento es el mismo que el dueño de la clave pública que acaba de aplicar para descifrar el resumen recibido.

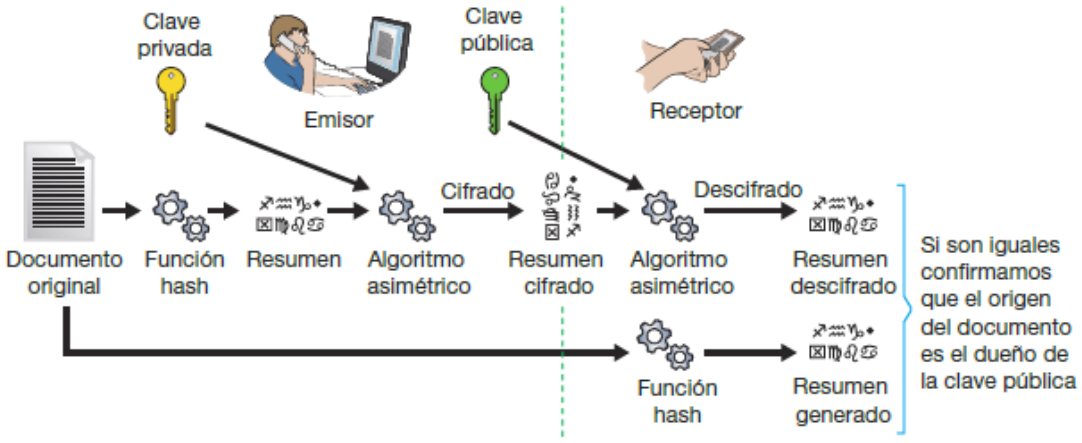


Figura 4: Mecanismo de firma.

Fuente: (ROA BUENDÍA, 2013)

Por supuesto, si queremos que el documento original no pueda ser interceptado en la transmisión desde el emisor al receptor, debemos cifrarlo. Para ello usaremos la clave pública del receptor. El procedimiento completo sería:

- El emisor aplica la función hash al original para generar el resumen.
- El emisor toma **su clave privada** para aplicar el algoritmo asimétrico al documento resumen. El resultado es un documento resumen cifrado.
- El emisor toma **la clave pública del receptor** para aplicar el algoritmo asimétrico al documento original y al documento resumen. El resultado es un documento conjunto cifrado que se envía al receptor.

En el receptor, utiliza **su clave privada** para descifrar los documentos y **la clave pública del origen** para comprobar la firma.

El mecanismo de firma también se utiliza en las comunicaciones de datos para garantizar al servidor que somos un cliente de confianza, y así podemos

evitar introducir usuario y contraseña **(autenticación sin contraseña).** Previamente, el servidor debe tener almacenada la clave pública del cliente (que habrá llegado hasta allí de manera segura).

Cuando el cliente empieza una sesión y solicita autenticación sin contraseña, el servidor genera en ese momento un documento especial, llamado **desafío,** compuesto de cifras y letras elegidas aleatoriamente. Busca en sus ficheros la clave pública del cliente, cifra con ella ese desafío y se lo envía al cliente.

El cliente lo recibe, lo intenta descifrar con su clave privada y el resultado lo devuelve al servidor. Entonces el servidor compara la secuencia de caracteres recibida con el desafío que generó; si son iguales, efectivamente el cliente es de confianza y puede conectar directamente. Todo este diálogo puede quedar a salvo de miradas ajenas si usamos una conexión cifrada mediante la clave pública del servidor.

2.2.5. CIFRADO POR BLOQUES

Los algoritmos de cifrado por bloques toman bloques de tamaño fijo del texto en claro y producen un bloque de tamaño fijo de texto cifrado (TALENS-OLIAG, 2003), generalmente del mismo tamaño que la entrada. El tamaño del bloque debe ser lo suficientemente grande como para evitar ataques de texto cifrado. La asignación de bloques de entrada a bloques de salida debe ser uno a uno para hacer el proceso reversible y parecer aleatoria.

Para la asignación de bloques los algoritmos de cifrado simétrico realizan sustituciones y permutaciones en el texto en claro hasta obtener el texto cifrado.

2.2.5.1. SUSTITUCIÓN

La *sustitución* es el reemplazo de un valor de entrada por otro de los posibles valores de salida, en general, si usamos un tamaño de bloque k, el bloque de entrada puede ser sustituido por cualquiera de los 2^k bloques posibles (TALENS-OLIAG, 2003).

Históricamente este tipo de cifrado se ha utilizado muchas veces, y es una buena ilustración de la criptografía básica. Vamos a utilizar el cifrado de sustitución para el aprendizaje de algunos hechos importantes sobre longitudes de clave y sobre las diferentes formas de atacar sistemas de cifrado.

El objetivo del cifrado de sustitución es el cifrado del texto (en oposición a los bits en los sistemas digitales modernos). La idea es muy simple: sustituimos cada letra del alfabeto por otra. (Paar & Pelzl, 2010)

2.2.5.2. PERMUTACIÓN

La *permutación* es un tipo especial de sustitución en el que los bits de un bloque de entrada son reordenados para producir el bloque cifrado, de este modo se preservan las estadísticas del bloque de entrada (el número de unos y ceros) (TALENS-OLIAG, 2003).

2.2.5.3. ALGORITMOS DE CIFRADO POR BLOQUES ITERATIVOS

Los algoritmos de cifrado por bloques *iterativos* funcionan aplicando en sucesivas rotaciones una transformación (función de rotación) a un bloque de texto en claro. La misma función es aplicada a los datos usando una subclave obtenida de la clave secreta proporcionada por el usuario. El número de

rotaciones en un algoritmo de cifrado por bloques iterativo depende del nivel de seguridad deseado.

Un tipo especial de algoritmos de cifrado por bloques iterativos son los denominados algoritmos de cifrado de Feistel. En estos algoritmos el texto cifrado se obtiene del texto en claro aplicando repetidamente la misma transformación o función de rotación. El funcionamiento es como sigue: el texto a cifrar se divide en dos mitades, la función de rotación se aplica a una mitad usando una subclave y la salida de la función se emplea para hacer una o-exclusiva con la otra mitad, entonces se intercambian las mitades y se repite la misma operación hasta la última rotación, en la que no hay intercambio. Una característica interesante de estos algoritmos es que la cifrado y descifrado es idénticas estructuralmente, aunque las subclaves empleadas en el cifrado se toman en orden inverso en el descifrado. (TALENS-OLIAG, 2003)

Para aplicar un algoritmo por bloques es necesario descomponer el texto de entrada en bloques de tamaño fijo. Esto se puede hacer de varias maneras:

1. ECB (Electronic Code Book). Se parte el mensaje en bloques de k bits, rellenando el ultimo si es necesario y se encripta cada bloque. para descifrar se trocea el texto cifrado en bloques de k bits y se desencripta cada bloque. Este sistema es vulnerable a ataques ya que dos bloques idénticos de la entrada generan el mismo bloque de salida. En la práctica no se utiliza
2. CBC (Cipher Block Chaining). Este método soluciona el problema del ECB haciendo una o-exclusiva de cada bloque de texto en claro con el bloque anterior cifrado antes de cifrar. para el primer bloque se usa un

vector de inicialización. Este es uno de los esquemas más empleados en la práctica.

3. OFB (Output Feedback Mode). Este sistema emplea la clave de la sesión para crear un bloque pseudoaleatorio grande (pad) que se aplica en o-exclusiva al texto en claro para generar el texto cifrado. Este método tiene la ventaja de que el pad puede ser generado independientemente del texto en claro, lo que incrementa la velocidad de cifrado y descifrado.
4. CFB (Cipher Feedback Mode). Variante del método anterior para mensajes muy largos.

2.2.6. AES (ADVANCED ENCRYPTION STANDAR)

AES (Advanced Encryption Standar) es el nuevo estándar de criptografía simétrica adoptado en el FIPS 197 (Federal Information Processing Standards). (ANGEL ANGEL, 2005) En este reporte estamos comprometidos a poder dar de la manera más simple la descripción total del algoritmo, y dar algunas características de importancia.

Desde 1977 que apareció la primera versión del estándar FIPS 46, asume como estándar el algoritmo DES (Data Encryption Standar), y sus posteriores reafirmaciones en 1983, 1988, 1993, y 1999. Casi siempre había visto opiniones controversiales de DES, sin embargo nunca fue dado un ataque que derivara por completo la clave secreta partiendo de la información pública, pero su corta longitud de clave lo comprometía poco a poco. La última reafirmación de DES en octubre de 1999 realmente fue suplantado por TDES, que es una versión múltiple

de DES, designado como TDEA (Triple Data Encryption Algorithm). De hecho, ya se tenían planes de encontrar un reemplazo definitivo a DES. A pesar de un número grande de algoritmos que en la época estaban presentes como: IDEA, RC5, skipjack, 3-way, FEAL, LOKI, SAFER, SHARK,... NIST decidió convocar a un concurso que tuvo como principales objetivos obtener un algoritmo simétrico que garantice su seguridad para los próximos 20 años a partir del año 2000. La convocatoria apareció el 2 de enero de 1997, se admitieron 15 algoritmos, en agosto de 1998 en la primera conferencia AES se discutieron los algoritmos sometidos y posteriormente en la segunda conferencia AES en marzo de 1999, se realizaron los últimos comentarios. Para que en agosto de 1999 se comunicaran los 5 finalistas: MARS, RC6, Rijndael, Serpent y Twofish. En abril del 2000 se llevó a cabo la tercera conferencia AES, recibiendo los últimos análisis, para que finalmente el 2 de octubre del año 2000 se diera a conocer el ganador y se dispuso al Algoritmo RIJNDAEL como AES. Esto llego a ser asumido oficial en noviembre 26 el 2001 en el FIPS 197. A partir de esa fecha hay conferencias especiales para analizar la situación actual de AES, la última se llevada a cabo en mayo del 2004.

El algoritmo Rijndael fue elegido principalmente por garantizar seguridad, que significa ser inmune a los ataques conocidos, tener un diseño simple, y poder ser implementado en la mayoría de los escenarios posibles, desde dispositivos con recursos limitados, como smart cards, hasta procesadores paralelos. El tiempo ha permitido que AES sea adaptado poco a poco, desde los protocolos más usados como SSL, hasta las aplicaciones más especializadas, como VoIP.

La descripción de AES es simple si se cuentan con todos los elementos. Esta consiste en dos partes, (ANGEL ANGEL, 2005) la primera en el proceso de cifrado y la segunda en el proceso de generación de las subclaves, una primera aproximación se muestra la siguiente figura:

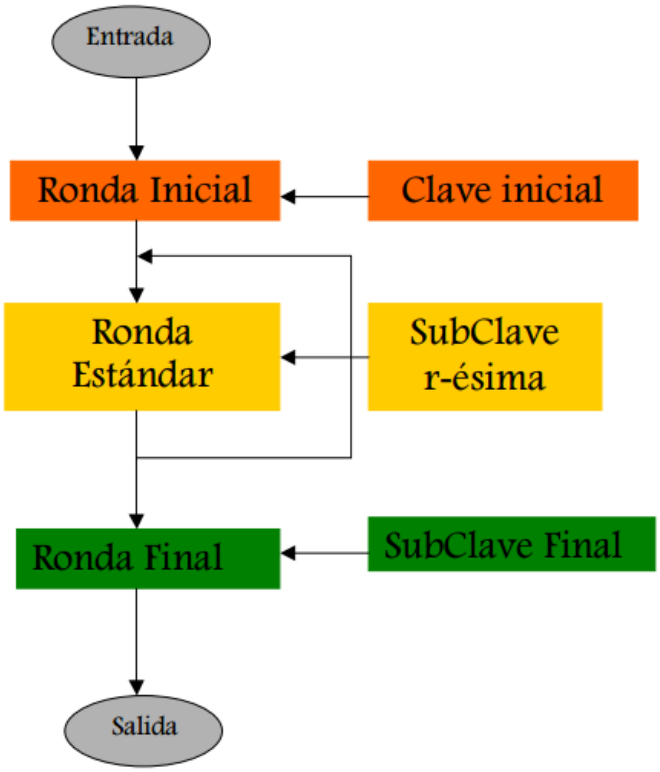


Figura 5: Descripción simple de AES

Fuente: (ANGEL ANGEL, 2005)

De manera un poco más detallada el algoritmo AES llega a ser:

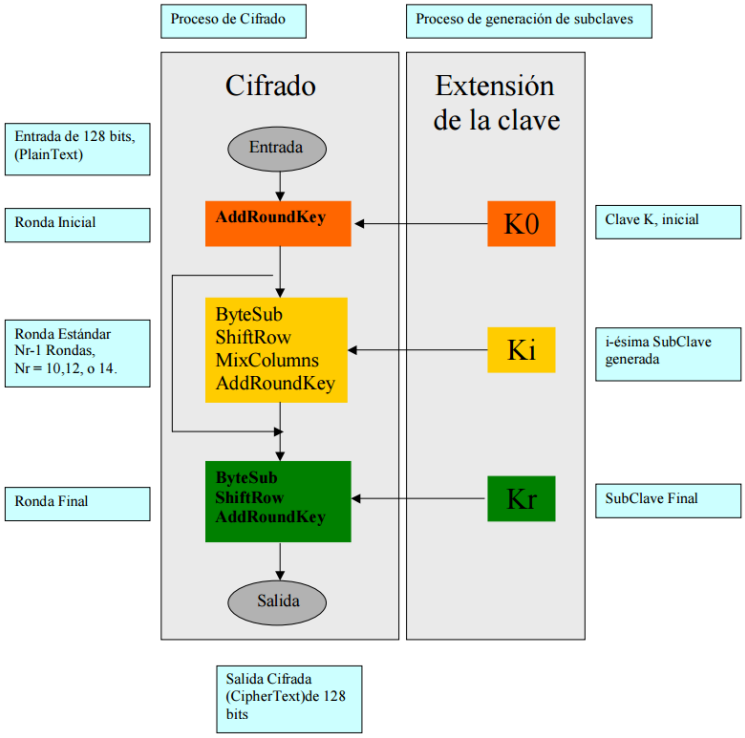


Figura 6: Descripción detallada de AES

Fuente: (ANGEL ANGEL, 2005)

Entonces la descripción de AES consiste de dos partes, en describir el proceso de “Cifrado”, y el proceso de “Generación de las subclaves” o “Extensión de la clave K”. El bloque de cifrado tiene una longitud de 128 bits, la longitud de la clave K varia de 128, 192 y 256 bits, en cada caso AES tiene 10,12, y 14 rondas respectivamente.

El proceso de cifrado consiste esencialmente en la descripción de las 4 transformaciones básicas de AES: ByteSub, ShiftRow, MixColumns, y AddRoundKey. Es importante mencionar que el caso de Rijndael las funciones o transformaciones básicas son ligeramente diferentes en el proceso de descifrado, sin embargo es poco el esfuerzo necesario para poder comprender todo.

2.2.6.1. EL CAMPO FINITO GF (2^8)

Es muy importante para poder entender el funcionamiento de AES, ya que AES trabaja como elementos básicos a los bytes (conjunto de 8 bits), (ANGEL ANGEL, 2005) AES ve a los bytes como elementos del campo finito GF (2^8), nuestro propósito es explicar que significa hacer esto.

Comenzamos recordando lo siguiente:

Los conjuntos de números más conocidos son: los números enteros Z, los números racionales Q, los números reales R, y los números complejos C.

Los números enteros Z, son los números enteros positivos y negativos Z = {... -2, -1, 0, 1,2,...}, los números racionales se construyen anexando los inversos multiplicativos que no hay en Z, así la definición de los números racionales queda como:

$$\mathbb{Q} = \left\{ \frac{a}{b} \middle| a, b \in \mathbb{Z}, b \neq 0 \right\}$$

Los racionales tienen una excelente propiedad algebraica, llamada estructura de campo, concretamente quiere decir que tanto con la operación suma, como la operación producto, tiene las siguientes propiedades: (ANGEL ANGEL, 2005)

$a + b = b + a$	Conmutatividad	$a \times b = b \times a$
$a + (b + c) = (a + b) + c$	Asociatividad	$a \times (b \times c) = (a \times b) \times c$
$a + 0 = a$	Existe el neutro	$a \times 1 = a$
$a + (-a) = 0$	Existe el inverso	$a\, x \frac{1}{a} = 1$

Tabla 1: Propiedades números racionales

Fuente: (ANGEL ANGEL, 2005)

Además de una propiedad más que une las dos operaciones y es la distributiva a× (b+c)=a×b+a×c.

De los conocimientos básicos sabemos que tanto el conjunto de los números Reales R, como los complejos C, también cumplen estas propiedades. Esto quiero decir que q, c, r son campos.

Otros conjuntos de números diferentes a los anteriores forman un campo, estudiados en matemáticas son, por ejemplo el conjunto de números de la forma {a, b ∈ R | a+b $\sqrt{-D}$}, para D adecuados también son un campo, de hecho podemos observar que si D =1, entonces tenemos a los complejos. (ANGEL ANGEL, 2005)

En la práctica se puede observar que dotar a un conjunto de la suma y verificar sus propiedades no es muy difícil y en muchos casos lo más complicado es poder mostrar que los elementos diferentes de cero tienen inverso multiplicativo dado un producto, de hecho en la criptografía esto es una propiedad muy recurrente.

Los ejemplos anteriores son usados en una amplia gama de aplicaciones, particularmente en la criptografía, sin embargo en nuestro caso, el caso de AES, lo que nos ocupa es el conocer el campos finitos GF (2^8). Por lo tanto enseguida daremos los elementos para poder entender como surgen los campos finitos, para esto demos las siguientes definiciones preliminares. (ANGEL ANGEL, 2005)

Recordemos como se define el conjunto Z_n que es el conjunto de residuos módulo n, es decir $\mathbb{Z}_n = \{[a] | 0 \leq a < n\}$, y $[a] = \{b | \exists s \in \mathbb{Z}\}$, hagamos algunos Ejemplos:

$\mathbb{Z}_2 = \{[0], [1]\}$, en este caso [0] son todos los números enteros donde existe s, tal que b = s2, es decir la división entre 2 deja un residuo cero, otra manera de decirlo son todos los números pares. [1] son los números b tales que existe un s y b = s2+1, es decir los números que dejan como residuo el 1, u otra forma, son los números impares. (ANGEL ANGEL, 2005)

Z_3= {[0], [1], [2]}, [0] son todos los números b tales que existe un s, y b = 3s, es decir los múltiplos de 3, o dejan residuo 0 al dividirlo por 3.

[1], son los números b tales que dejan residuo 1 al dividirlos por 3.

[2], son los números b tales que dejan residuo 2 al dividirlos por 4.

Z_4= {[0], [1], [2], [3]}, [0] son los números b tales que existe un s, y b=4s.

[1], son los números b que dejan residuo 1 al dividirlos por 4.

[2], son los números b que dejan residuo 2 al dividirlos por 4.

[3], son los números b que dejan residuo 3 al dividirlos por 4.

Por razones de simplificación omitiremos la notación [_], dando por entendido que en n **Z**, se trabaja con clases y no con números simples.

Otra manera de ver los elementos de n **Z**, es colocarlos de la siguiente forma, por ejemplo para $\mathbf{Z}_5$

-5 -4 -3 -2 -1

0 1 2 3 4

5 6 7 8 9

Del anterior ejemplo podemos inferir que –1 en 5 $\mathbf{Z_5}$ es 4, en general –1 en n $\mathbf{Z_n}$ es n-1, -2=n-2,...

De manera natural podemos dotar a $\mathbf{Z_n}$ de las operaciones de suma y producto, [a] + [b] = [a + b], [a] [b] = [ab].

Ahora podemos preguntarnos cuándo $\mathbf{Z_n}$ es campo, sin mucho problema podemos chequear para que n's $\mathbf{Z_n}$ es campo, y se deriva de los siguientes hechos:

1. Es fácil ver que ($\mathbf{Z_n}$,+) satisfacen las primeras propiedades de campo, se dice que ($\mathbf{Z_n}$,+) es un grupo abeliano.
2. ($\mathbf{Z_n}$, x) es fácil ver que es asociativo, que es conmutativo, que existe la identidad, sin embargo no siempre existen los inversos multiplicativos.

En el caso de que existan los inversos multiplicativos en n $\mathbf{Z_n}$, entonces $\mathbf{Z_n}$ será un campo, por lo que nos toca investigar cuando siempre existen los inversos y esto se deduce del siguiente hecho:

Teorema: (Algoritmo de Euclides) sean a, n, números enteros, entonces siempre existen números enteros q, r tales que n =qa+r.

El anterior lema puede ser mostrado de manera inmediata.

Algoritmo extendido de Euclides: dados dos números enteros a, n, y d su máximo común divisor, entonces siempre existen s, t tales que d = sa + tn.

Del anterior teorema, si a es primo relativo a n, entonces su máximo común divisor es el 1, entonces del Teorema extendido de Euclides, existen números s, t tales que 1 = sa+t *n*, al aplicar la clase de equivalencia modulo n, tenemos [1] = [s] [a], ya que [n] = 0, es decir que. $[S]= [a]^{-1}$

Lo anterior significa que un número a en Zn tiene inverso multiplicativo si y sólo si, a es primo relativo a n.

Se sigue entonces que Zn es campo sí y sólo si, n es número primo, ya que en este caso todos los números 0≤a<n son primos relativos a n, y por lo tanto tienen inverso multiplicativo. (ANGEL ANGEL, 2005)

Ejemplos:

Z_2 es campo, y sus operaciones son las siguientes:

+	0	1
0	0	1
1	1	0

●	0	1
0	0	0
1	0	1

Tabla 2: Operaciones para el campo Z_2

Fuente: (ANGEL ANGEL, 2005)

Sin problema podemos ver también que Z_3, Z_5, Z_7 son campos.

Para Z_3= {0,1, 2}, el inverso de 2 es el propio 2.

Para Z_5= {5 0,1, 2,3, 4}, el inverso de 2 es el 3, el inverso de 4 es el mismo 4.

Para Z_7= {7 0,1, 2,3, 4,5, 6}, el inverso de 2 es 4, el inverso de 3 es 5, el de 6 es el mismo 6.

De la misma manera Z_4, Z_6 no son campos ya que:

Para Z_4= {4 0,1, 2,3}, el 2 no tiene inverso multiplicativo.

Para Z_6= {0,1, 2, 3, 4,5}, el 2 tampoco tiene inverso multiplicativo.

En resumen los primeros ejemplos de campos finitos son de la forma Z_p, con p número primo.

En seguida damos un teorema que nos indica que forma tiene todos los campos finitos, y de ahí podremos hablar del campo finito que nos interesa para AES. (ANGEL ANGEL, 2005)

Afirmación: todo campo finito tiene p^n elementos, y se puede construir a partir de Z_p.

Observe que la inversa no es cierta, es decir, si existe un conjunto con p^n elementos no necesariamente es campo, como el caso de Z_4.

Veamos ahora algunas definiciones y resultados que nos ayudará a construir un campo finito.

Definición: sea F un campo, entonces F[x] es el conjunto de polinomios con indeterminada x.

Podemos definir de manera convencional en F[x] las operaciones de suma y producto. De hecho F[x] es un anillo conmutativo con identidad, es decir, solo le falta que todos sus elementos tengan inverso para ser campo.

Definición: sea f(x) un polinomio en F[x], entonces f es irreducible sobre F, si f tiene grado positivo y si f=gh, g, h en F[x], entonces g o h son constantes.

Ejemplos

Sea 2 $f \in Z_2[x]$, con $f(x)=x^5+x^3+x+1$ es un polinomio con indeterminadas en x y coeficientes en Z_2.

2.2.6.2. CONSTRUCCIÓN DE CAMPO FINITO GF (2^8)

Una de las cosas más importantes de AES es conocer muy bien el campo GF (2^8), así como sus operaciones y su representación. (ANGEL ANGEL, 2005) Sabemos del anterior punto que es necesario encontrar un polinomio irreducible con coeficientes en F= Z_2 de grado 8.

Antes de continuar observemos que cualquier polinomio $f \in Z_2[x]$ se puede ver como una lista de bits, de esa manera AES toma por convenio el siguiente orden:

(10001111) ~ $x^7 + x^3 + x^2 + x + 1$

(11001100) ~ $x^7 + x^6 + x^3 + x^2$

(10101010) ~ $x^7 + x^5 + x^3 + x$

Es decir

Si $p(x) \in GF(2^8)$, entonces $p(x)=b_7x^7+b_6x^6+b_5x^5+b_4x^4+b_3x^3+b_2x^2+b_1x+b_0$ corresponde al byte (b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0) a continuación listamos todos los polinomios irreducibles hasta grado 8, los últimos nos servirán para la construcción de $GF(2^8)$.

Polinomios irreducibles de grado 2

Binario	Hex	Polinomio	Orden
111	7	$x^2 + x + 1$	3

Tabla 3: Polinomios irreducibles de grado 2

Fuente: (ANGEL ANGEL, 2005)

Polinomios irreducibles de grado 3

Binario	Hex	Polinomio	Orden
1011	B	$x^3 + x + 1$	7
1101	D	$x^3 + x^2 + 1$	7

Tabla 4: Polinomios irreducibles de grado 3

Fuente: (ANGEL ANGEL, 2005)

Polinomios irreducibles de grado 4

Binario	Hex	Polinomio	Orden
10011	13	x^4+x+1	15
11001	19	x^4+x^3+1	15
11111	1f	$x^4+x^3+x^2+x+1$	5

Tabla 5: Polinomios irreducibles de grado 4

Fuente: (ANGEL ANGEL, 2005)

Polinomios irreducibles de grado 5

Binario	Hex	Polinomio	Orden
100101	25	x^5+x^2+1	31
101001	29	x^5+x^3+1	31
101111	2f	$x^5+x^3+x^2+x+1$	31
110111	37	$x^5+x^4+x^2+x+1$	31
111011	3b	$x^5+x^4+x^3+x+1$	31
111101	3d	$x^5+x^4+x^3+x^2+1$	31

Tabla 6: Polinomios irreducibles de grado 5

Fuente: (ANGEL ANGEL, 2005)

Polinomios irreducibles de grado 6

Binario	Hex	Polinomio	Orden
1000011	43	x^6+x+1	63
1001001	49	x^6+x^3+1	9
1010111	57	$x^6+x^4+x^2+x+1$	21

1011011	5b	$x^6 + x^4 + x^3 + x + 1$	63
1100001	61	$x^6 + x^5 + 1$	63
1100111	67	$x^6 + x^5 + x^2 + x + 1$	63
1101101	6d	$x^6 + x^5 + x^3 + x^2 + 1$	63
1110011	73	$x^6 + x^5 + x^4 + x + 1$	63
1110101	75	$x^6 + x^5 + x^4 + x^2 + 1$	21

Tabla 7: Polinomios irreducibles de grado 6

Fuente: (ANGEL ANGEL, 2005)

Polinomios irreducibles de grado 7

Binario	Hex	Polinomio	Orden
10000011	83	$x^7 + x + 1$	127
10001001	89	$x^7 + x^3 + 1$	127
10001111	8f	$x^7 + x^3 + x^2 + x + 1$	127
10010001	91	$x^7 + x^4 + 1$	127
10011101	9d	$x^7 + x^4 + x^3 + x^2 + 1$	127
10100111	a7	$x^7 + x^5 + x^2 + x + 1$	127
10101011	Ab	$x^7 + x^5 + x^3 + x + 1$	127
10111001	b9	$x^7 + x^5 + x^4 + x^3 + 1$	127
10111111	Bf	$x^7 + x^5 + x^4 + x^3 + x^2 + x + 1$	127
11000001	c1	$x^7 + x^6 + 1$	127
11001011	Cb	$x^7 + x^6 + x^3 + x + 1$	127
11010011	d3	$x^7 + x^6 + x^4 + x + 1$	127
11010101	d5	$x^7 + x^6 + x^4 + x^2 + 1$	127

11100101	e5	$x^7 +x^6 +x^5 +x^2 +1$	127
11101111	Ef	$x^7 +x^6 +x^5 +x^3 +x^2 +x+1$	127
11110001	f1	$x^7 +x^6 +x^5 +x^4 +1$	127
11110111	f7	$x^7 +x^6 +x^5 +x^4 +x^2 +x+1$	127
11111101	Fd	$x^7 +x^6 +x^5 +x^4 +x^3 +x^2 +1$	127

Tabla 8: Polinomios irreducibles de grado 7

Fuente: (ANGEL ANGEL, 2005)

Polinomios irreducibles de grado 8

Binario	Hex	Polinomio	Orden
100011011	11b	$x^8 +x^4+x^3 +x+1$	51
100011101	11d	$x^8 +x^4+x^3 +x^2 +1$	255
100101011	12b	$x^8 +x^5+x^3 +x+1$	255
100101101	12d	$x^8 +x^5+x^3 +x^2+1$	255
100111001	139	$x^8 +x^5+x^4 +x^3+1$	17
100111111	13f	$x^8 +x^5+x^4 +x^3+x^2 +x+1$	85
101001101	14d	$_x8+x^6+x^3+x^2+1$	255
101011111	15f	$x^8+x^6+x^4 +x^3+x^2 +x+1$	255
101100011	163	$x^8+x^6+x^5+x+1$	255
101100101	165	$x^8+x^6+x^5+x^2+1$	255
110110001	1b1	$x^8 +x^7 +x^5+x^4 +1$	51
110111101	1bd	$x^8 +x^7 +x^5+x^4 +x^3 +x^2+1$	85
111000011	1c3	$x^8 +x^7 +x^6+x+1$	255
111001111	1cf	$x^8 +x^7 +x^6+x^3 +x^2+x+1$	255

111010111	1d7	$x^8+x^7+x^6+x^4+x^2+x+1$	17
111011101	1dd	$x^8+x^7+x^6+x^4+x^3+x^2+1$	85
111100111	1e7	$x^8+x^7+x^6+x^5+x^2+x+1$	255
111110011	1f3	$x^8+x^7+x^6+x^5+x^4+x+1$	51
111110101	1f5	$x^8+x^7+x^6+x^5+x^4+x^2+1$	255
111111001	1f9	$x^8+x^7+x^6+x^5+x^4+x^3+1$	85

Tabla 9: Polinomios irreducibles de grado 8

Fuente: (ANGEL ANGEL, 2005)

Ahora tenemos todos los elementos para poder construir al campo GF (2^8), usado en AES. Antes se explicaran los datos que se dieron en las listas anteriores, particularmente el orden del polinomio irreducible.

Como sabemos el campo GF(2^8) es el cociente $Z_2[x]/(f)$ donde f es cualquier polinomio de los arriba escritos, en unas hojas más se mostrara que en el caso especial de AES es indiferente que polinomio se elija, sin embargo para ser compatibles AES toma al primer polinomio irreducible, es decir $m(x)=x^8+x^4+x^3+x+1$.

Como recordamos $Z^2[x]/(m(x))$ es el conjunto de residuos modulo m(x), es decir el conjunto de polinomios de grado menor a 8. Las operaciones de campo a partir del polinomio m(x) se obtienen modulo m(x), es decir, es el residuo de la división entre m(x).

Para ejemplificar veamos la construcción campos de menor grado:

La construcción del campo finito de 2 elementos, GF(2^2), para esto tomemos el polinomio f(x)=x^2 +x+1 con coeficientes en Z_2 , este polinomio es irreducible, ahora sea α una raíz, entonces α^2=1+α , de aquí observamos que:

$$\alpha^0 = 1$$

$$\alpha^1 = \alpha$$

$$\alpha^2 = 1 + \alpha$$

$$\alpha^3 = \alpha(1 + \alpha) = \alpha + \alpha^2 = 1$$

Se dice que α genera a un grupo cíclico de orden 2^2 -1=3 elementos.

Entonces el campo finito GF (2^2), de 2^2 elementos tiene como tablas de suma y multiplicación a

+	0	1	A	1+α
0	0	1	A	1+α
1	1	0	1+α	A
A	A	1+α	0	1
1+α	1+α	A	1	0

●	1	α	1+α
1	1	α	1+α
A	α	1+α	1
1+α	1+α	1	α

Tabla 10: Tablas de suma y multiplicación del campo finito $GF(2^2)$

Fuente: (ANGEL ANGEL, 2005)

Cualquier elemento de GF(2^2), es un polinomio de grado menor a 2, y coeficientes en $\mathbf{Z}_2$, de otra forma un elemento de GF(2^2), es un elemento del cociente $\mathbf{Z}_2$ [x]/(x^2 +x+1), o más bien un elemento de ahí es una clase de equivalencia que consiste de todos los polinomios que al dividirlos por f(x) su residuo es un polinomio de grado menor a 2, de la misma manera como los elementos de $\mathbf{Z}_p$

Para construir ahora el campo GF(2^3), es necesario tomar un polinomio irreducible y entonces una raíz de ese polinomio tiene orden según lo que se lista, en este caso los dos polinomios son de orden 7, es decir la elección de cualquiera de los dos, una raíz nos generara todo el campo. De la misma manera, cualquier polinomio f(x) con coeficientes en $\mathbf{Z}_2$, pertenece a una clase, y se sabe a que clase pertenece cuando se obtiene el residuo de f(x) con el polinomio irreducible. Observamos también que todos los residuos serán todos los polinomios de grado menor a 3, es decir los polinomios de grado a lo más 2.

En este caso tenemos dos polinomios irreducibles hagamos las operaciones que corresponden a cada caso y veamos algunas de sus diferencias.

$m(x) = x^3 + x + 1$, entonces:

$$x^0 = 1$$

$$x^1 = x$$

$$x^2 = x2$$

$$x^3 = x + 1$$

$$x^4 = x^2 + x$$

$$x^5 = x^2 + x + 1$$

$$x^6 = x^2 + 1$$

$$x^7 = 1$$

$m(x) = x^3 + x^2 + 1$, entonces

$$x^0 = 1$$

$$x^1 = x$$

$$x^2 = x2$$

$$x^3 = x^2 + 1$$

$$x^4 = x^2 + x + 1$$

$$x^5 = x + 1$$

$$x^6 = x^2 + x$$

$$x^7 = 1$$

Para el primer caso la tabla de producto queda de la siguiente manera

●	**x**	**x+1**	**x^2**	**x^2+1**	**x^2+x**	**x^2+x+1**
x	x^2	x^2 + x	x+1	1	x^2+x+1	x^2+1
x+1	x^2 + x	x^2 + 1	x^2 + x+1	x^2	1	X
x^2	x+1	x^2 + x+1	x^2 + x	X	x^2+1	1
x^2+1	1	x^2	X	x^2+x+1	x+1	x^2+x
x^2+x	x^2+x+1	1	x^2+1	x+1	X	x^2
x^2+x+1	x^2+1	x^2+x	1	x^2+x	x^2	x+1

Tabla 11: Operación producto para el primer caso

Fuente: (ANGEL ANGEL, 2005)

Para el segundo caso tenemos

●	x	x+1	x^2	x^2+1	x^2+x	x^2+x+1
X	x^2	x^2+x	x^2+1	x^2+x+1	1	x+1
x+1	x^2+x	x^2+1	1	x	x^2+x+1	x2
x^2	x^2+1	1	x^2+x+1	x+1	x	x^2+x
x^2+1	x^2+x+1	x	x+1	x^2+x	x2	1
x^2+x	1	x^2+x+1	x	x^2	x+1	x^2+1
x^2+x+1	x+1	x^2	x^2+x	1	x^2+1	x^2+x

Tabla 12: Operación producto para el segundo caso

Fuente: (ANGEL ANGEL, 2005)

Observamos lo siguiente:

1. Tanto en un caso como en otro los polinomios involucrados sólo cambian de lugar en las dos tablas de multiplicación.
2. Sin embargo hay dos cosa que considerar, la primera en los dos casos el polinomio es irreducible, por lo tanto x es un generador y así el producto se puede realizar con la formula $x^i x^j \square x^{(i+j) \bmod 2^n - 1}$, que pueden ser usadas para el producto siempre y cuando el polinomio sea primitivo. Segundo donde hay diferencia en el tiempo de obtener el producto si depende del polinomio irreducible y es al obtener el residuo, esto se nota más en campos grandes, donde la obtención de un producto consiste en hacer la división por respectivos polinomios irreducibles tardando más en el caso de que este polinomio sea más grande, en tales casos es preferible buscar polinomios irreducibles con el menor número de términos.

3. Existen otras formas de representar este tipo de campos que no son el propósito de este reporte, por ejemplo usando bases normales.

Hay que tener cuidado solo en el caso donde el polinomio es irreducible y su grado (el dato de la derecha en la tabla de polinomios) no alcanza a generar todo el grupo multiplicativo, en estos casos debemos elegir otro generador, el siguiente teorema dice que siempre podemos encontrar un generador, más aún hay suficientes de estos para no tener problemas.

Teorema: Todo campo finito tiene un generador. Si g es un generador de F_q^* , entonces g^i es también un generador si y silo si (i, q-1)=1, es decir hay un total de $\phi(q-1)$ generadores diferentes.

El caso de F_{24} tenemos un polinomio irreducible no primitivo (1f), es decir, que x no genera al grupo cíclico multiplicativo. Dependiendo del caso que tengamos podemos elegir un polinomio primitivo o no.

Nota importante: para el caso de AES no existe diferencia en seguridad si el polinomio es o no primitivo, más aun, se verá en la sección 8 que no es importante la elección del polinomio.

En los siguientes casos la construcción del campo es similar a los anteriores.

Particularmente el campo GF(2^8), se construye tomando el primer polinomio irreducible de la lista $m(x)= x^8+x^4+x^3+x+1$ (11b) que tiene orden 51, es decir x no genera a todo el campo, sin embargo en el proceso de las multiplicaciones se toma al polinomio g(x)=x+1 como generador.

Nota importante: AES toma como base al campo GF(2^8), ya que considera a un byte(conjunto de 8 bits) como su palabra básica, haciendo posible así, la implementación en muchas plataformas a partir de los 8 bits.

El campo GF(2^8) entonces se muestra en la siguiente tabla con todos sus elementos, donde el entero es la potencia i de $(1+x)^i$ y el polinomio el resultado de $(1+x)^i$ mod m(x):

Lista de todos los elementos del campo GF(2^8) usando el polinomio irreducible $m(x) = x^8 + x^4 + x^3 + x + 1$

1. 1+x
2. $1+ x^2$
3. $1+ x + x^2 + x^3$
4. $1+ x^4$
5. $1+ x + x^4 + x^5$
6. $1+ x^2 + x^4 + x^6$
7. $1+ x + x^2 + x^3 + x^4 + x^5 + x^6 + x^7$
8. $x+ x^3 + x^4$
9. $x+ x^2 + x^3 + x^5$
10. $x+ x^4 + x^5 + x^6$
11. $x+ x^2 + x^4 + x^7$
12. $1+ x^5 + x^7$
13. $x^3 + x^4 + x^5 + x^6 + x^7$
14. $1+ x + x^4$
15. $1+ x^2 + x^4 + x^5$
16. $1+ x + x^2 + x^3 + x^4 + x^6$
17. $1+ x^5 + x^6 + x^7$
18. $x^3 + x^4 + x^5$
19. $x^3 + x^6$
20. $x^3 + x^4 + x^6 + x^7$
21. $1+ x + x^4 + x^5 + x^6$
22. $1+ x^2 + x^4 + x^7$
23. $x^2 + x^5 + x^7$
24. $1+ x + x^2 + x^4 + x^5 + x^6 + x^7$
25. x
26. $x+ x^2$
27. $x+ x^3$
28. $x+ x^2 + x^3 + x^4$
29. $x+ x^5$
30. $x+ x^2 + x^5 + x^6$

31. $x+ x^3 + x^5 +x^7$
32. $1+ x^2 + x^5 +x^6 + x^7$
33. $x^2 + x^4 + x^5$
34. $x^2 + x^3 + x^4 + x^6$
35. $x^2 + x^5 + x^6 + x^7$
36. $1+ x + x^2 + x^4 + x^5$
37. $1+ x^3 + x^4 +x^6$
38. $1+ x + x^3 + x^5 + x^6 + x^7$
39. $x+ x^2 + x^5$
40. $x+ x^3 + x^5 +x^6$
41. $x+ x^2 + x^3 +x^4 + x^5 + x^7$
42. $1+ x^3 + x^4 +x^6 + x^7$
43. $x^4 + x^5 + x^6$
44. $x^4 + x^7$
45. $1+ x + x^3 + x^5 + x^7$
46. $x+ x^2 + x^5 +x^6 + x^7$
47. $1+ x^4 + x^5$
48. $1+ x + x^4 + x^6$
49. $1+ x^2 + x^4 +x^5 + x^6 + x^7$
50. x^2
51. $x^2 + x^3$
52. $x^2 + x^4$
53. $x^2 + x^3 + x^4 + x^5$
54. $x^2 + x^6$
55. $x^2 + x^3 + x^6 + x^7$
56. $1+ x + x^2 + x^3 + x^6$
57. $1+ x^4 + x^6 +x^7$
58. $x^3 + x^5 + x^6$
59. $x^3 + x^4 + x^5 + x^7$
60. $1+ x + x^4 + x^6 + x^7$
61. $x+ x^2 + x^3 +x^5 + x^6$
62. $x+ x^4 + x^5 +x^7$
63. $1+ x^2 + x^3 +x^6 + x^7$
64. $x^2 + x^3 + x^6$
65. $x^2 + x^4 + x^6 + x^7$
66. $1+ x + x^2 + x^5 + x^6$
67. $1+ x^3 + x^5 +x^7$
68. $x^5 + x^6 + x^7$
69. $1+ x + x^3 + x^4 + x^5$
70. $1+ x^2 + x^3 +x^6$
71. $1+ x + x^2 + x^4 + x^6 + x^7$
72. $x+ x^5 + x^6$
73. $x+ x^2 + x^5 +x^7$
74. $1+ x^4 + x^5 +x^6 + x^7$
75. x^3
76. $x^3 + x^4$
77. $x^3 + x^5$
78. $x^3 + x^4 + x^5 + x^6$

79. $x^3 + x^7$

80. $1+ x + x^7$

81. $x+ x^2 + x^3 +x^4 + x^7$

82. $1+ x^3 + x^4 +x^5 + x^7$

83. $x^4 + x^6 + x^7$

84. $1+ x + x^3 + x^5 + x^6$

85. $1+ x^2 + x^3 +x^4 + x^5 + x^7$

86. $x^2 + x^3 + x^4 + x^6 + x^7$

87. $1+ x + x^2 + x^3 + x^4 + x^5 + x^6$

88. $1+ x^7$

89. $x^3 + x^4 + x^7$

90. $1+ x + x^4 + x^5 + x^7$

91. $x+ x^2 + x^3 +x^6 + x^7$

92. $1+ x^3 + x^6$

93. $1+ x + x^3 + x^4 + x^6 + x^7$

94. $+ x^2 + x^4 +x^5 + x^6$

95. $x+ x^3 + x^4 +x^7$

96. $1+ x^2 + x^4 +x^5 + x^7$

97. $x^2 + x^6 + x^7$

98. $1+ x + x^2 + x^4 + x^6$

99. $1+ x^3 + x^4 +x^5 + x^6 + x^7$

100. x^4

101. $x^4 + x^5$

102. $x^4 + x^6$

103. $x^4 + x^5 + x^6 + x^7$

104. $1+ x + x^3$

105. $1+ x^2+ x^3 +x^4$

106. $1+ x + x^2 + x^5$

107. $1+ x^3+ x^5 +x^6$

108. $1+ x + x^3 + x^4 + x^5 + x^7$

109. $x+ x^2+ x^4 +x^6 + x^7$

110. $1+ x^5+ x^6$

111. $1+ x + x^5 + x^7$

112. $x+ x^2+ x^3 +x^4 + x^5 + x^6 + x^7$

113. $1+ x^3+ x^4$

114. $1+ x + x^3 + x^5$

115. $1+ x^2+ x^3 +x^4 + x^5 + x^6$

116. $1+ x + x^2 + x^7$

117. $x+ x^4+ x^7$

118. $1+ x^2+ x^3 +x^5 + x^7$

119. $x^2 + x^3 + x^5 + x^6 + x^7$

120. $1+ x + x^2 + x^3 + x^5$

121. $1+ x^4 + x^5 +x^6$

122. $1 + x + x^4 + x^7$

123. $x+ x^2+ x^3 +x^5 + x^7$

124. $1 + x^3+ x^5 +x^6 + x^7$

125. x^5

126. $x^5 + x^6$

127. $x^5 + x^7$

128. $1 + x + x^3 + x^4 + x^5 + x^6 + x^7$

129. $x+ x^2 + x^4$

130. $x+ x^3 + x^4 + x^5$

131. $x+ x^2+ x^3 + x^6$

132. $x+ x^4+ x^6 + x^7$

133. $1 + x^2 + x^3 + x^5 + x^6$

134. $1 + x + x^2 + x^4 + x^5 + x^7$

135. $x+ x^6 + x^7$

136. $1 + x^2 + x^3 + x^4 + x^6$

137. $1 + x + x^2 + x^5 + x^6 + x^7$

138. $x+ x^4+ x^5$

139. $x+ x^2 + x^4 + x^6$

140. $x+ x^3 + x^4 + x^5 + x^6 + x^7$

141. $1 + x^2 + x^4$

142. $1+ x + x^2 + x^3 + x^4 + x^5$

143. $1 + x^6$

144. $1 + x + x^6 + x^7$

145. $x+ x^2+ x^3 + x^4 + x^6$

146. $x+ x^5+ x^6 + x^7$

147. $1+ x^2+ x^3 + x^4 + x^5$

148. $1+ x + x^2 + x^6$

149. $1+ x^3 + x^6 + x^7$

150. x^6

151. $x^6 + x^7$

152. $1+ x + x^3 + x^4 + x^6$

153. $1+ x^2+ x^3 + x^5 + x^6 + x^7$

154. $x^2 + x^3 + x^5$

155. $x^2 + x^4 + x^5 + x^6$

156. $x^2 + x^3 + x^4 + x^7$

157. $1+ x + x^2 + x^3 + x^4 + x^5 + x^7$

158. $x+ x^3+ x^4 + x^6 + x^7$

159. $1+ x^2+ x^4 + x^5 + x^6$

160. $1+ x + x^2 + x^3 + x^4 + x^7$

161. $x+ x^3+ x^4 + x^5 + x^7$

162. $1+ x^2+ x^4 + x^6 + x^7$

163. $x^2 + x^5 + x^6$

164. $x^2 + x^3 + x^5 + x^7$

165. $1+ x + x^2 + x^3 + x^5 + x^6 + x^7$

166. $x+ x^3+ x$

167. $x+ x^2 + x^3 + x^4 + x^5 + x^6$

168. $x+ x^7$

169. $1+ x^2+ x^3 + x^4 + x^7$

170. $x^2 + x^3 + x^4 + x^5 + x^7$

171. $1+ x + x^2 + x^3 + x^4 + x^6 + x^7$

172. $x+ x^3+ x^4 + x^5 + x^6$

173. $x+ x^2+ x^3 + x^7$

174. $1+ x^3+ x^7$

175. x^7

176. $1+ x + x^3 + x^4 + x^7$

177. $x+ x^2+ x^4 +x^5 + x^7$

178. $1+ x^6 + x^7$

179. $x^3 + x^4 + x^6$

180. $x^3 + x^5 + x^6 + x^7$

181. $1+ x + x^5$

182. $1 + x^2+ x^5 +x^6$

183. $1+ x + x^2 + x^3 + x^5 + x^7$

184. $x+ x^3 + x^5 +x^6 + x^7$

185. $1+ x^2+ x^5$

186. $1+ x + x^2 + x^3 + x^5 + x^6$

187. $1+ x^4+ x^5 +x^7$

188. $x^3 + x^6 + x^7$

189. $1+ x + x^6$

190. $1 + x^2+ x^6 +x^7$

191. $x^2 + x^4 + x^6$

192. $x^2 + x^3 + x^4 + x^5 + x^6 + x^7$

193. $1 + x + x^2 + x^3 + x^4$

194. $1 + x^5$

195. $1 + x + x^5 + x^6$

196. $1 + x^2 + x^5 +x^7$

197. $x^2 + x^4 + x^5 + x^6 + x^7$

198. $1+ x + x^2$

199. $1+ x^3$

200. $1+ x + x^3 + x^4$

201. $1+ x^2 + x^3 +x^5$

202. $1+ x + x^2 + x^4 + x^5 + x^6$

203. $1+ x^3+ x^4 +x^7$

204. $x^4 + x^5 + x^7$

205. $1+ x + x^3 + x^6 + x^7$

206. $x+ x^2 + x^6$

207. $x+ x^3 + x^6 +x^7$

208. $1+ x^2+ x^6$

209. $1+ x + x^2 + x^3 + x^6 + x^7$

210. $x+ x^3+ x^6$

211. $x+ x^2+ x^3 +x^4 + x^6 + x^7$

212. $1+ x^3+ x^4 +x^5 + x^6$

213. $1+ x + x^3 + x^7$

214. $x+ x^2+ x^7$

215. $1+ x^4 + x^7$

216. $x^3 + x^5 + x^7$

217. $1 + x + x^5 + x^6 + x^7$

218. $x+ x^2 + x^3 +x^4 + x^5$

219. $x+ x^6$

220. $x+ x^2+ x^6 +x^7$

221. $1+ x^4+ x^6$

222. $1+ x + x^4 + x^5 + x^6 + x^7$

223. $x+ x^2+ x^3$

224. $x+ x^4$

225. $x+ x^2+ x^4 +x^5$

226. $x+ x^3+ x^4 +x^6$

227. $x+ x^2+ x^3 +x^5 + x^6 + x^7$

228. $1+ x^3+ x^5$

229. $1+ x + x^3 + x^4 + x^5 + x^6$

230. $1+ x^2+ x^3 +x^7$

231. $x^2 + x^3 + x^7$

232. $1+ x + x^2 + x^3 + x^7$

233. $x+ x^3+ x^7$

234. $1+ x^2+ x^7$

235. $x^2 + x^4 + x^7$

236. $1+ x + x^2 + x^5 + x^7$

237. $x+ x^4+ x^5 +x^6 + x^7$

238. $1+ x^2+ x^3$

239. $1+ x + x^2 + x^4$

240. $1+ x^3+ x^4 +x^5$

241. $1+ x + x^3 + x^6$

242. $1+ x^2+ x^3 +x^4 + x^6 + x^7$

243. $x^2 + x^3 + x^4 + x^5 + x^6$

244. $x^2 + x^7$

245. $1+ x + x^2 + x^4 + x^7$

246. $x+ x^5+ x^7$

247. $1+ x^2+ x^3 +x^4 + x^5 + x^6 + x^7$

248. $x^2 + x^3 + x^4$

249. $x^2 + x^5$

250. $x^2 + x^3 + x^5 + x^6$

251. $x^2 + x^4 + x^5 + x^7$

252. $1+ x + x^2 + x^6 + x^7$

253. $x+ x^4 + x^6$

254. $x+ x^2 + x^4 +x^5 + x^6 + x^7$

255. 1

256. $1+x$

Para terminar con la representación del campo $GF(2^8)$, damos las siguientes dos reglas que son usadas en el código para poder multiplicar dos elementos del campo $GF(2^8)$. La idea es simple, si queremos multiplicar dos elementos digamos $a,b \in GF(2^8)$, entonces existen i,j tales que $a=(1+x)^i$, $b=(1+x)^j$ por lo tanto $ab=(1+x)^{i+j \bmod 255}$, si basta encontrar el elemento que esta la

posición (i+j)mod255 para saber el resultado. En términos de notación conocida decimos que ab=AntiLog((Log[a]+Log[b]) mod 255), próximamente veremos en el código que usa esta fórmula para calcular los productos.

2.2.6.3. EL ANILLO GF(2^8) [X]/(X^4 +1).

Otra estructura que usa AES es la del anillo GF(2^8) [x]/(x^4 +1), otra de las operaciones básicas de AES es multiplicar un polinomio de tercer grado con coeficientes en GF(2^8) , (ANGEL ANGEL, 2005) por una constante. Es decir, un elemento de GF(2^8) [x]/(x^4 +1), por un polinomio constante, el resultado se reduce modulo (x^4 +1) para obtener un polinomio de grado 3. El objetivo de lo anterior es poder definir multiplicación columnas de 4 bytes por un elemento constante también de 4 bytes, en el proceso de descifrado se aplica la operación inversa y aunque el polinomio (x^4 +1) no es irreducible, en este caso particular la constante si tiene inverso en el anillo GF(2^8) [x]/(x^4 +1), por lo tanto la constante tiene inverso.

Sea a(x) un polinomio tal que $a_0 + a_1x + a_2x^2 + a_3x^3 \in Gf(2^8)[x]/(x^4 + 1)$, donde $a_i \in GF(2^8)$ y b(x) otro polinomio igual, entonces a(x)b(x)=c(x) donde c(x) tiene la misma forma, particularmente:

$$a(x)b(x)a0 = (a_0 + a_1x + a_2x^2 + a_3x^3)(b_0 + b_1x + b_2x^2 + b_3x^3)$$

$$= a0\,b0 + (a1\,b0 + a0\,b1)x + (a2\,b0 + a1\,b1 + a0\,b2)x2$$

$$+(a_3b_0 + a_2b_1 + a_1b_2 + a0\,b3)x3$$

$$+(a_3b_1 + a_2b_2 + a1\,b3)x4$$

$$+(a_3b_2 + a_2b_3)x^5 + a_3b_3x^6$$

El siguiente paso es aplicar modulo x^4+1 a todo el polinomio, que es aplicar modulo a cada uno de sus términos, entonces $x^i \bmod (x^4+1) = x^{i \bmod 4}$, entonces el resultado de a(x)b(x) queda como

$$\begin{aligned} a(x)b(x) = {} & (a_0b_0 + a_3b_1 + a_2b_2 + a_1b_3) \\ & +(a_1b_0 + a_0b_1 + a_3b_2 + a_2b_3)x \\ & +(a_2b_0 + a_1b_1 + a_3b_2 + a_2b_3)x^2 \\ & +(a_3b_0 + a_2b_1 + a_1b_2 + a_0b_3)x^3 \\ & = c_0 + c_1x + c_2x^2 + c_3x^3 \end{aligned}$$

Finalmente de manera matricial el anterior producto puede ser visto como:

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

2.2.7. DES (DATA ENCRYPTION STANDARD)

Se trata de un sistema de cifrado simétrico por bloques de 64 bits, de los que 8 bits (un byte) se utilizan como control de paridad (para la verificación de la integridad de la clave). Cada uno de los bits de la clave de paridad (1 cada 8 bits) se utiliza para controlar uno de los bytes de la clave por paridad impar, es decir, que cada uno de los bits de paridad se ajusta para que tenga un número impar de "1" dentro del byte al que pertenece. Por lo tanto, la clave tiene una longitud "útil" de 56 bits, es decir, realmente sólo se utilizan 56 bits en el algoritmo. (CCM.net - Kioskea ES, 2015)

El algoritmo se encarga de realizar combinaciones, sustituciones y permutaciones entre el texto a cifrar y la clave, asegurándose al mismo tiempo de que las operaciones puedan realizarse en ambas direcciones (para el

descifrado). La combinación entre sustituciones y permutaciones se llama cifrado del producto.

La clave es codificada en 64 bits y se compone de 16 bloques de 4 bits, generalmente anotadas dek_1 a k_{16}. Dado que "solamente" 56 bits sirven para el cifrado, ¡puede haber hasta 2^{56} (o $7.2*10^{16}$) claves diferentes

2.2.7.1. EL ALGORITMO DES

Las partes principales del algoritmo son las siguientes:

- fraccionamiento del texto en bloques de 64 bits (8 bytes),
- permutación inicial de los bloques,
- partición de los bloques en dos partes: izquierda y derecha, denominadas *I* y *D*respectivamente,
- fases de permutación y de sustitución repetidas 16 veces (denominadas **rondas**),
- reconexión de las partes izquierda y derecha, seguida de la permutación inicial inversa. (CCM.net - Kioskea ES, 2015)

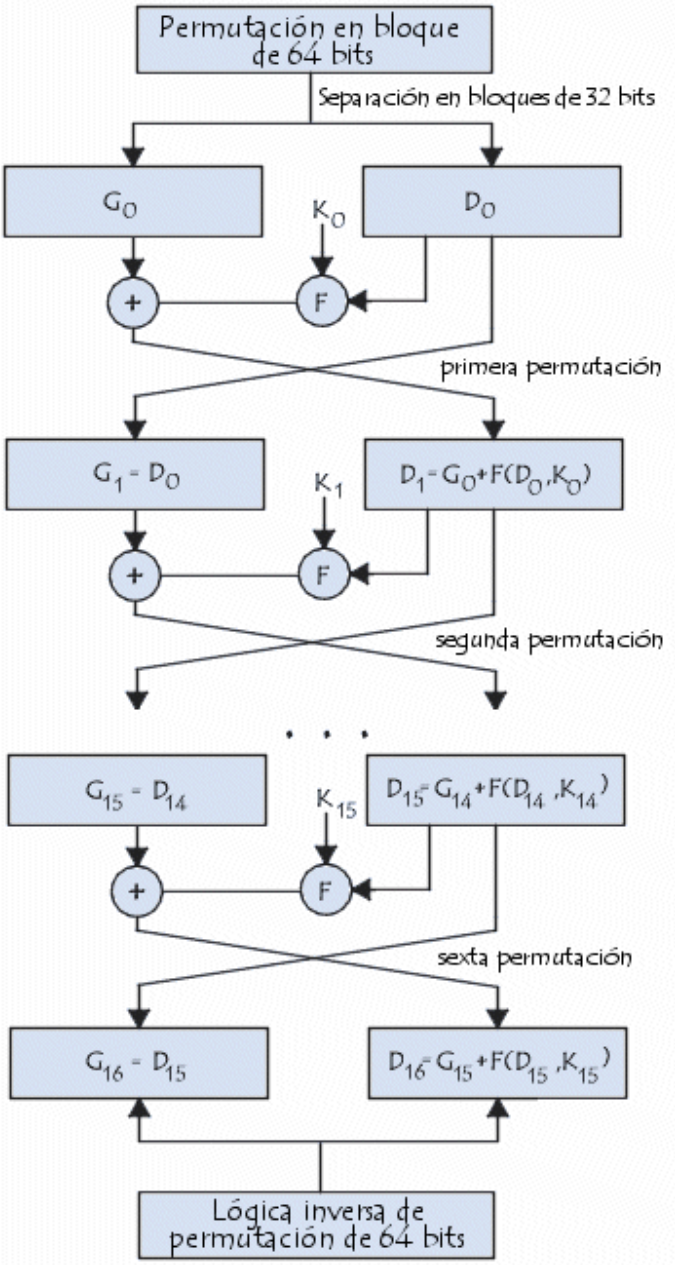


Figura 7: Algoritmo DES

Fuente: (CCM.net - Kioskea ES, 2015)

2.2.7.2. PERMUTACIÓN INICIAL

En primer lugar, cada bit de un bloque está sujeto a una permutación inicial, que puede representarse mediante la siguiente matriz de permutación inicial (*anotada como PI*): (CCM.net - Kioskea ES, 2015)

IP								
	58	50	42	34	26	18	10	2
	60	52	44	36	28	20	12	4
	62	54	46	38	30	22	14	6
	64	56	48	40	32	24	16	8
	57	49	41	33	25	17	9	1
	59	51	43	35	27	19	11	3
	61	53	45	37	29	21	13	5
	63	55	47	39	31	23	15	7

Tabla 13: Matriz de permutación inicial

Fuente: (CCM.net - Kioskea ES, 2015)

Esta tabla de permutación muestra, al leerla de izquierda a derecha y de arriba a abajo, que el 58ºbit de un bloque de 64 bits está en la primera posición, el 50º está en la segunda posición y así sucesivamente

2.2.7.3. DIVISIÓN EN BLOQUES DE 32 BITS

Una vez que la permutación inicial se completó, el bloque de 64 bits se divide en dos bloques de 32 bits denominados **I** y **D** respectivamente (para izquierda y derecha, siendo la anotación en anglo-sajón L y R por Left y Right). El estado inicial de estos dos bloques se denomina $\mathbf{L}_0$ y $\mathbf{R}_0$: (CCM.net - Kioskea ES, 2015)

L_0								
	58	50	42	34	26	18	10	2
	60	52	44	36	28	20	12	4
	62	54	46	38	30	22	14	6
	64	56	48	40	32	24	16	8

Tabla 14: División de bloques Bloque Left

Fuente: (CCM.net - Kioskea ES, 2015)

R_0								
	57	49	41	33	25	17	9	1
	59	51	43	35	27	19	11	3
	61	53	45	37	29	21	13	5
	63	55	47	39	31	23	15	7

Tabla 15: División de bloques Bloque Right

Fuente: (CCM.net - Kioskea ES, 2015)

Es interesante observar que L_0 contiene todos los bits que se encuentran en posición par en el mensaje inicial, mientas que R_0 contiene los bits en posición impar.

2.2.7.4. RONDAS

Los bloques L_n y R_n están sujetos a un conjunto de transformaciones iterativas denominadas *rondas*, que se muestran en este esquema y que detallamos a continuación: (CCM.net - Kioskea ES, 2015)

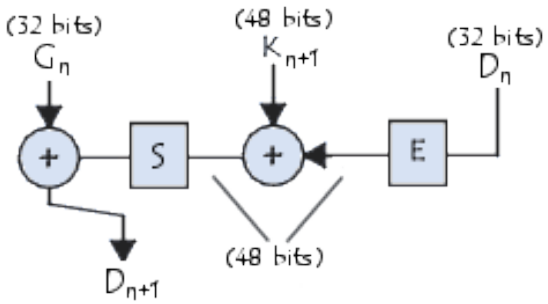


Figura 8: Rondas o Transformaciones Iterativas

Fuente: (CCM.net - Kioskea ES, 2015)

2.2.7.5. FUNCIÓN DE EXPANSIÓN

Los 32 bits del bloque $\mathbf{R}_0$ se expanden a 48 bits gracias a una tabla (matriz) llamada *tabla de expansión* (que se anota como **E**), en la que los 48 bits se mezclan y 16 de ellos se duplican: (CCM.net - Kioskea ES, 2015)

E						
	32	1	2	3	4	5
	4	5	6	7	8	9
	8	9	10	11	12	13
	12	13	14	15	16	17
	16	17	18	19	20	21
	20	21	22	23	24	25
	24	25	26	27	28	29
	28	29	30	31	32	1

Tabla 16: Tabla de expansión

Fuente: (CCM.net - Kioskea ES, 2015)

Así, el último bit de $\mathbf{R}_0$ (es decir, el 7º bit del bloque de origen) se convierte en el primero, el primero en el segundo, etc.

Además, los bits 1, 4, 5, 8, 9, 12, 13, 16, 17, 20, 21, 24, 25, 28 y 29 de $\mathbf{R}_0$ (respectivamente los bits 57, 33, 25,l , 59, 35,27 , 3, 6l, 37, 29, 5, 63, 39, 31 y 7 del bloque de origen) son duplicados y diseminados en la matriz.

2.2.7.6. OR EXCLUSIVA CON LA CLAVE

La tabla resultante de 48 bits se denomina $\mathbf{D'}_0$ o $\mathbf{E}[\mathbf{D}_0]$. El algoritmo DES aplica después *OR exclusivas* entre la primera clave $\mathbf{K}_1$ y $\mathbf{E}[\mathbf{D}_0]$. El resultado de este *OR exclusivo* es una tabla de 48 bits que, por comodidad, (CCM.net - Kioskea ES, 2015) llamaremos $\mathbf{D}_0$

2.2.7.7. FUNCIÓN DE SUSTITUCIÓN

Después, $\mathbf{D}_0$ se divide en 8 bloques de 6 bits, denominado $\mathbf{D}_{0i}$. Cada uno de estos bloques se procesa a través de **funciones de selección** (a veces llamadas *cajas de sustitución* o *funciones de compresión*), denominadas generalmente $\mathbf{S}_i$. (CCM.net - Kioskea ES, 2015)

Los primeros y últimos bits de cada $\mathbf{D}_{0i}$ determinan (en valor binario) la línea de la función de selección; los otros bits (2, 3, 4 y 5 respectivamente) determinan la columna. Como la selección de la línea se basa en dos bits, existen 4 posibilidades (0,1,2,3). Como la selección de la columna se basa en 4 bits, existen 16 posibilidades (0 a 15). Gracias a esta información, la función de selección "selecciona" un valor cifrado de 4 bits.

Esta es la primera función de sustitución, representada en una tabla de 4 por 16: (CCM.net - Kioskea ES, 2015)

$\mathbf{S}_1$

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

Tabla 17: Primera función de sustitución

Fuente: (CCM.net - Kioskea ES, 2015)

Sea $\mathbf{R}_{01}$ igual a *101110*. El primer y último bit dan *10*, es decir, 2 en valor binario. Los bits 2,3,4 y 5 dan *0111*, o 7 en valor binario. Por lo tanto, el resultado de la función de selección es el valor ubicado en la línea nº 2, de la columna nº 7. Es el valor *11* o *111* en binario.

Cada uno de los 8 bloques de 6 bits pasa a través de la función de selección correspondiente, dando un resultado de 8 valores con 4 bits cada uno. A continuación están las otras funciones de selección:

S_2

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
2	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
3	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

Tabla 18: Segunda tabla de Función de sustitución

Fuente: (CCM.net - Kioskea ES, 2015)

S_3

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
2	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
3	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

Tabla 19: Tercera tabla de Función de sustitución

Fuente: (CCM.net - Kioskea ES, 2015)

S_4

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
2	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

Tabla 20: Cuarta tabla de Función de sustitución

Fuente: (CCM.net - Kioskea ES, 2015)

S_5

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
2	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
3	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

Tabla 21: Quinta tabla de Función de sustitución

Fuente: (CCM.net - Kioskea ES, 2015)

S_6

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
2	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
3	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

Tabla 22: Sexta tabla de Función de sustitución

Fuente: (CCM.net - Kioskea ES, 2015)

S_7

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
1	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
2	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
3	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

Tabla 23: Séptima tabla de Función de sustitución

Fuente: (CCM.net - Kioskea ES, 2015)

S_8																
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
1	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
1	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Tabla 24: Octava tabla de Función de sustitución

Fuente: (CCM.net - Kioskea ES, 2015)

Por lo tanto, cada bloque de 6 bits se sustituye por un bloque de 4 bits. Estos bits se combinan para formar un bloque de 32 bits. (CCM.net - Kioskea ES, 2015)

2.2.7.8. PERMUTACIÓN

Finalmente, el bloque de 32 bits se somete a una permutación **P**. A continuación, mostramos la tabla: (CCM.net - Kioskea ES, 2015)

P							
16	7	20	21	29	12	28	17
1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9
19	13	30	6	22	11	4	25

Tabla 25: Tabla de permutación

Fuente: (CCM.net - Kioskea ES, 2015)

2.2.7.1. OR EXCLUSIVO

El conjunto de estos resultados salidos de **P** están sujetos a un *OR exclusivo* con $\mathbf{I}_0$ inicial (como se muestra en el primer esquema) para devolver D1, en tanto que la $\mathbf{D}_0$ inicial devuelve $\mathbf{I}_1$. (CCM.net - Kioskea ES, 2015)

2.2.7.1. ITERACIÓN

El conjunto de los pasos anteriores (*rondas*) se reitera 16 veces. (CCM.net - Kioskea ES, 2015)

2.2.7.1. PERMUTACIÓN INICIAL INVERSA

Al final de las iteraciones, los dos bloques $\mathbf{L}_{16}$ y $\mathbf{R}_{16}$ se vuelven a conectar y se someten a una permutación inicial inversa: (CCM.net - Kioskea ES, 2015)

IP-1								
	40	8	48	16	56	24	64	32
	39	7	47	15	55	23	63	31
	38	6	46	14	54	22	62	30
	37	5	45	13	53	21	61	29
	36	4	44	12	52	20	60	28
	35	3	43	11	51	19	59	27
	34	2	42	10	50	18	58	26
	33	1	41	9	49	17	57	25

Tabla 26: Permutación inicial inversa

Fuente: (CCM.net - Kioskea ES, 2015)

El resultado que surge es un texto cifrado de 64 bits

2.2.7.1. GENERACIÓN DE CLAVES

Dado que el algoritmo DES mencionado anteriormente es público, toda la seguridad se basa en la complejidad de las claves de cifrado.

El algoritmo que sigue a continuación muestra cómo obtener a partir una clave de 64 bits (compuesta por cualquier de los 64 caracteres alfanuméricos), 8 claves diferentes de 48 bits, cada una de ellas utilizadas en el algoritmo DES: (CCM.net - Kioskea ES, 2015)

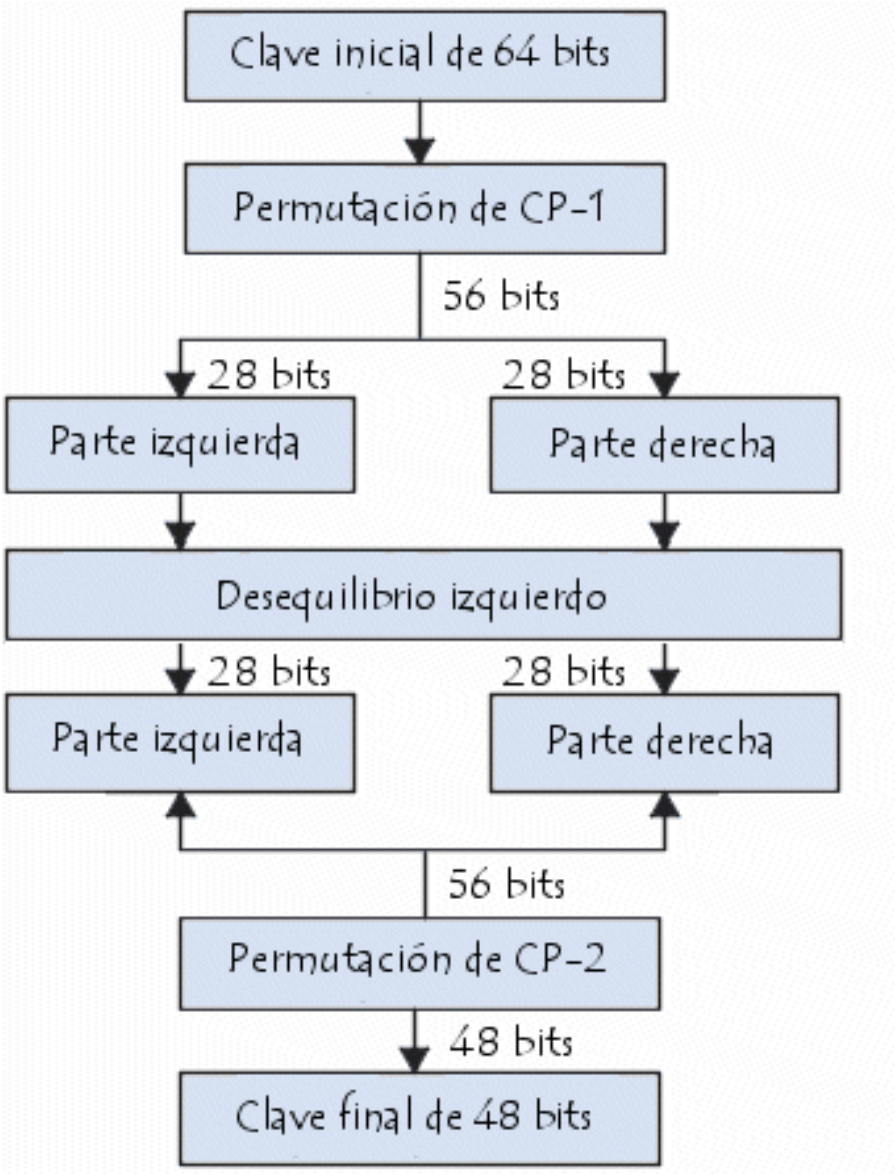


Figura 9: Generación de claves de 64 bits

Fuente: (CCM.net - Kioskea ES, 2015)

En primera instancia, se eliminan los bits de paridad de la clave para obtener una clave que posea una longitud de 56 bits.

El primer paso es una permutación denominada **PC-1**, cuya tabla se presentará a continuación: (CCM.net - Kioskea ES, 2015)

PC-1														
	57	49	41	33	25	17	9	1	58	50	42	34	26	18
	10	2	59	51	43	35	27	19	11	3	60	52	44	36
	63	55	47	39	31	23	15	7	62	54	46	38	30	22
	14	6	61	53	45	37	29	21	13	5	28	20	12	4

Tabla 27: Permutación PC-1

Fuente: (CCM.net - Kioskea ES, 2015)

Esta matriz puede escribirse en forma de dos matrices $\mathbf{L}_i$ y $\mathbf{R}_i$ (para la izquierda y la derecha respectivamente), cada una ellas de 28 bits:

$\mathbf{L}_i$							
	57	49	41	33	25	17	9
	1	58	50	42	34	26	18
	10	2	59	51	43	35	27
	19	11	3	60	52	44	36

Tabla 28: Matriz Li

Fuente: (CCM.net - Kioskea ES, 2015)

$\mathbf{R}_i$							
	63	55	47	39	31	23	15
	7	62	54	46	38	30	22
	14	6	61	53	45	37	29
	21	13	5	28	20	12	4

Tabla 29: Matriz Ri

Fuente: (CCM.net - Kioskea ES, 2015)

El resultado de esta primera permutación se denomina $\mathbf{I}_0$ y $\mathbf{D}_0$.

Luego, estos dos bloques se rotan hacia la izquierda, de manera que los bits que estaban en la segunda posición pasan a la primera, aquellos que estaban en tercera posición pasan a la segunda, etc.

Los bits que estaban en la primera posición se mueven hacia la última posición.

Los dos bloques de 28 bits se agrupan en un bloque de 56 bits. Este pasa por una permutación, denominada **PC-2**, (CCM.net - Kioskea ES, 2015) dando como resultado un bloque de 48 bits que representa la clave $\mathbf{K}_i$.

pc-2												
	14	17	11	24	1	5	3	28	15	6	21	10
	23	19	12	4	26	8	16	7	27	20	13	2
	41	52	31	37	47	55	30	40	51	45	33	48
	44	49	39	56	34	53	46	42	50	36	29	32

Tabla 30: Permutación PC-2

Fuente: (CCM.net - Kioskea ES, 2015)

Realizando iteraciones del algoritmo es posible obtener las 16 claves $\mathbf{K}_1$ a $\mathbf{K}_{16}$ utilizadas en un algoritmo DES.

LS																
	1	2	4	6	8	10	12	14	15	17	19	21	23	25	27	28

Tabla 31: 16 claves utilizadas en el Algoritmo DES

Fuente: (CCM.net - Kioskea ES, 2015)

2.2.1. ANÁLISIS DE ALGORITMOS

El análisis de algoritmos nos permite medir la dificultad inherente de un problema y evaluar la eficiencia de un algoritmo. (VALENZUELA RUZ, 2003)

2.2.1.1. TIEMPOS DE EJECUCIÓN

Una medida que suele ser útil conocer es el tiempo de ejecución de un algoritmo en función de *N*, lo que denominaremos *T*(*N*). Esta función se puede medir físicamente (ejecutando el programa, reloj en mano), o calcularse sobre el código contando instrucciones a ejecutar y multiplicando por el tiempo requerido por cada instrucción. (VALENZUELA RUZ, 2003)

Así, un trozo sencillo de programa como:

$$S1;\ FOR\ i:=\ 1\ TO\ N\ DO\ S2\ END;$$

Requiere:

$$T(N):=\ t1\ +\ t2 * N$$

Siendo *t*1 el tiempo que lleve ejecutar la serie "*S*1" de sentencias, y *t*2 el que lleve la serie "*S*2".

Prácticamente todos los programas reales incluyen alguna sentencia condicional, haciendo que las sentencias efectivamente ejecutadas dependan de los datos concretos que se le presenten. (VALENZUELA RUZ, 2003) Esto hace que más que un valor *T*(*N*) debamos hablar de un rango de valores:

$$Tmin(N)\ <=\ T(N)\ <=\ Tmax(N)$$

Los extremos son habitualmente conocidos como "*caso peor*" y "*caso mejor*".

Entre ambos se hallará algún "*caso promedio*" o más frecuente. Cualquier fórmula *T*(*N*) incluye referencias al parámetro *N* y a una serie de constantes "T_i" que dependen de factores externos al algoritmo como pueden ser la calidad del código generado por el compilador y la velocidad de ejecución de instrucciones del computador que lo ejecuta.

2.2.1.1. CONCEPTO DE COMPLEJIDAD

La complejidad (o costo) de un algoritmo es una medida de la cantidad de recursos (tiempo, memoria) que el algoritmo necesita. La complejidad de un algoritmo se expresa en función del tamaño (o talla) del problema.

La función de complejidad tiene como variable independiente el tamaño del problema y sirve para medir la complejidad (espacial o temporal). Mide el tiempo/espacio relativo en función del tamaño del problema.

El comportamiento de la función determina la eficiencia. No es única para un algoritmo: depende de los datos. Para un mismo tamaño del problema, las distintas presentaciones iniciales de los datos dan lugar a distintas funciones de complejidad. Es el caso de una ordenación si los datos están todos inicialmente desordenados, parcialmente ordenados o en orden inverso.

2.2.1.1. NOTACIÓN ASINTÓTICA

La notación asintótica se describe por medio de una función cuyo dominio es los números naturales (N) estimado a partir de tiempo de ejecución o de espacio de memoria de algoritmos en base a la longitud de la entrada. Se consideran las funciones asintóticamente no negativas.

La notación asintótica captura el **comportamiento** de la función para valores grandes de *N*.

Las notaciones no son dependientes de los tres casos anteriormente vistos, es por eso que una notación que determine el peor caso puede estar presente en una o en todas las situaciones. (VALENZUELA RUZ, 2003)

2.2.1.2. LA O MAYÚSCULA

La notación O se utiliza para comparar funciones. Resulta particularmente útil cuando se quiere analizar la complejidad de un algoritmo, en otras palabras, la cantidad de tiempo que le toma a un computador ejecutar un programa.

Se utilizara en la investigación para analizar la complejidad temporal de las funciones del algoritmo Rijndael original y de su modificación. (VALENZUELA RUZ, 2003)

Definición: Sean f y g funciones con dominio en **R** ≤ 0 o **N** es imagen en R. Si existen constantes C y k tales que:

$$\forall\, x \;>\; k, |f\,(x)| \;\leq\; C\,|\,g\,(x)\,|$$

Es decir, que para x > k, f es menor o igual a un múltiplo de g, decimos que:

$$f\,(x) \;=\; O\,(\,g\,(x)\,)$$

La definición formal es:

$$f\,(x) \;=\; O\left(\,g\,(x)\right) \Leftrightarrow \exists\, k\,, N\,|\,\forall\, x \;>\; N, |\,f\,(x)\,| \;\leq\; k|\,g\,(x)\,|$$

¿Qué quiere decir todo esto? Básicamente, que una función es siempre menor que otra función (por ejemplo, el tiempo en ejecutar tu programa es menor que x2) si no tenemos en cuenta los factores constantes (eso es lo que significa la k) y si no tenemos en cuenta los valores pequeños (eso es lo que significa la N).

¿Por qué no tenemos en cuenta los valores pequeños de N? Porqué para entradas pequeñas, el tiempo que tarda el programa no es significativo y casi siempre el algoritmo será suficientemente rápido para lo que queremos. (VALENZUELA RUZ, 2003)

Así $3N^3 + 5N^2 - 9 = O\,(N^3)$ no significa que existe una función $O(N^3)$ que es igual a $3N^3 + 5N^2 - 9$.

Debe leerse como:

"$3N^3 + 5N^2 - 9$ es O-Grande de N^3"

Que significa:

"$3N^3 + 5N^2 - 9$ está **asintóticamente** dominada por N^3"

2.2.1.1. LA *o* MINÚSCULA

La cota superior asintótica dada por la notación ***O*** puede o no ser ajustada asintóticamente. La cota $2n^2 = \boldsymbol{O}(n^2)$ es ajustada asintóticamente, pero la cota $2n^2 = \boldsymbol{O}(n^2)$ no lo es. Utilizaremos la notación ***o*** para denotar una cota superior que no es ajustada asintóticamente. (VALENZUELA RUZ, 2003)

Definimos formalmente ***o***(*g(n)*) ("o pequeña") como el conjunto:

$\boldsymbol{o}(g(n)) = \{f(n)$: Para cualquier constante positiva *c* > 0, existe una constante $n_0 > 0$ tal que: $0 \leq f(n) \leq cg(n)$ para toda $n \geq n_0\}$.

Para *o* la desigualdad se mantiene para todas las constantes positivas, mientras que para *O* la desigualdad se mantiene sólo para algunas constantes positivas.

2.2.1.2. LAS NOTACIONES Ω Y Θ

Ω Es el reverso de *O*.

$$f(x) = \Omega(g(x)) \rightarrow\leftarrow g(x) = O(f(x))$$

Ω Grande dice que asintóticamente $f(x)$ *domina a* $g(x)$.

Θ Grande dice que ambas funciones se dominan mutuamente, en otras palabras, son asintóticamente equivalentes. (VALENZUELA RUZ, 2003)

$$f(x) = \Theta(g(x))$$

$$\rightarrow\leftarrow$$

$$f\,(x) = O\,(g\,(x)) \wedge f\,(x) = \Omega(g\,(x))$$

$f = \Theta(g)$: “f es de orden g”

2.2.2. ALGORITMO DE COMPRESIÓN DE HUFFMAN

Se trata de un algoritmo que puede ser usado para compresión o encriptación de datos.

Este algoritmo se basa en asignar códigos de distinta longitud de bits a cada uno de los caracteres de un fichero. Si se asignan códigos más cortos a los caracteres que aparecen más a menudo se consigue una compresión del fichero. Esta compresión es mayor cuando la variedad de caracteres diferentes que aparecen es menor. Por ejemplo: si el texto se compone únicamente de números o mayúsculas, se conseguirá una compresión mayor. (Coronado, 2001)

Para recuperar el fichero original es necesario conocer el código asignado a cada carácter, así como su longitud en bits, si ésta información se omite, y el receptor del fichero la conoce, podrá recuperar la información original. De este modo es posible utilizar el algoritmo para encriptar ficheros.

2.2.2.1. MECANISMO DEL ALGORITMO DE HUFFMAN

- Contar cuantas veces aparece cada carácter en el fichero a comprimir. Y crear una lista enlazada con la información de caracteres y frecuencias.
- Ordenar la lista de menor a mayor en función de la frecuencia.
- Convertir cada elemento de la lista en un árbol.

- Fusionar todos estos árboles en uno único, para hacerlo se sigue el siguiente proceso, mientras la lista de árboles contenga más de un elemento:
  - Con los dos primeros árboles formar un nuevo árbol, cada uno de los árboles originales en una rama.
  - Sumar las frecuencias de cada rama en el nuevo elemento árbol.
  - Insertar el nuevo árbol en el lugar adecuado de la lista según la suma de frecuencias obtenida.
- Para asignar el nuevo código binario de cada carácter sólo hay que seguir el camino adecuado a través del árbol. Si se toma una rama cero, se añade un cero al código, si se toma una rama uno, se añade un uno.
- Se recodifica el fichero según los nuevos códigos.

Basados en el esquema de Huffman existen algoritmos, como el LZH, ARJ, ZIP, LZW, CCITT group 3. (Menendez, 2009)

2.2.3. PRUEBA DE HIPÓTESIS ESTADÍSTICA

En todo trabajo de investigación, es necesario realiza una prueba confiable de hipótesis, que consiste en un procedimiento de decisión estadística que establece la metodología a seguir para la aceptación o rechazo de una hipótesis planteada sobre la base de las evidencias contenidas en un conjunto de observaciones muéstrales (Loayza, 2012)

2.2.3.1. PRUEBA T DE STUDENT PARA MUESTRAS RELACIONADAS

La prueba de t Student para muestras dependientes se utiliza para comparar las medias de un mismo grupo en diferentes etapas. Se utiliza, por ejemplo, para las comparaciones de los resultados de una prueba antes y después para un grupo determinado. (Gonzalez, 2008)

$$t_c = \frac{M_d}{DS_d/\sqrt{n}} \quad \text{.....eq(1)}$$

$$M_d = \sum_{i}^{n} \frac{(x_{2i} - x_{1i})}{n}$$

$$DS_d = \sqrt{\frac{\sum_1^n (d_i - \bar{x})^2}{n-1}}$$

Donde:

$$M_d = Media\ aritmetica\ de\ las\ diferencias$$

$$DS_d = Desviación\ estándar\ de\ las\ diferencias$$

$$n = Número\ de\ sujetos\ de\ la\ muestra$$

$$d_i = diferencias$$

2.2.3.2. INTERVALOS DE CONFIANZA DE LA DIFERENCIA ENTRE LAS MEDIAS DE DOS POBLACIONES NORMALES

Para comparar medias de dos poblaciones, se extraen muestras aleatorias de las dos poblaciones. El método que empleamos para seleccionar las muestras determina el método que debemos utilizar para analizar inferencias basadas en los resultados muéstrales (Newbold, Carlson, & Thorne, 2008)

MUESTRAS INDEPENDIENTES, VARIANZAS POBLACIONALES CONOCIDAS

En este sistema, se extraen muestras independientemente de las dos poblaciones que siguen una distribución normal y tienen varianzas poblacionales conocidas, por lo que en la pertenencia a una de las muestras no influye la pertenencia a la otra.

Consideremos el caso en el que se extraen de las dos poblaciones de interés muestras independientes, no necesariamente del mismo tamaño. Supongamos que tenemos una muestra aleatoria de n_x observaciones procedentes de una población de media μ_x y varianza σ_x^2 y una muestra aleatoria independiente de n_y observaciones procedentes de una población de media μ_y y varianza σ_y^2. Sean las medias muéstrales respectivas $\bar{x}$ e $\bar{y}$.

Examinemos, en primer lugar, la situación en la que las dos distribuciones poblacionales son normales y tienen varianzas conocidas. Como lo que nos interesa es la diferencia entre las dos medias poblacionales, es lógico basar una inferencia en la diferencia entre las medias muéstrales correspondientes. Esta variable aleatoria tiene una media

$$E(\bar{X} - \bar{Y}) = E(\bar{X}) - E(Y) = \mu_x - \mu_y$$

y como las muestras son independientes,

$$Var\,(\bar{X} - \bar{Y}) = Var\,(\bar{X}) + Var\,(\bar{Y}) = \frac{\sigma_x^2}{n_x} + \frac{\sigma_y^2}{n_y}$$

Puede demostrarse, además, que su distribución es normal. Se deduce, pues, que la variable aleatoria

$$Z = \frac{(\bar{x} - \bar{y}) - (\mu_x - \mu_y)}{\sqrt{\frac{\sigma_x^2}{n_x} + \frac{\sigma_y^2}{n_y}}} \quad \text{.....eq(2)}$$

Sigue una distribución normal estándar

Donde:

$$\bar{x} - \bar{y}: Diferencia\ de\ medias\ de\ las\ muestras.$$

$$\mu_x - \mu_y: Diferencias\ de\ medias\ poblacional.$$

$$\sigma_x: Desviación\ estándar\ de\ la\ primera\ muestra.$$

$$\sigma_y: Desviación\ estándar\ de\ la\ segunda\ muestra.$$

$$n_x: Tamaño\ de\ la\ primera\ muestra.$$

$$n_y: Tamaño\ de\ la\ segunda\ muestra.$$

2.3. GLOSARIO DE TÉRMINOS

Criptografía.- Arte y técnica de escribir con procedimientos o claves secretas o de un modo enigmático, de tal forma que lo escrito solamente sea inteligible para quien sepa descifrarlo.

Algoritmo.- Conjunto ordenado de operaciones sistemáticas que permite hacer un cálculo y hallar la solución de un tipo de problemas.

Cifrado.- Que está escrito con letras, símbolos o números que solo pueden comprenderse si se dispone de la clave necesaria para descifrarlos.

2.4. HIPÓTESIS DE LA INVESTIGACIÓN

2.4.1. HIPÓTESIS GENERAL

El nuevo algoritmo encriptación basado en el estándar de encriptación avanzada AES (Advanced Encryption Standard) ofrece una mayor independencia

matemática en os bloques crípticos y no se afectó la complejidad temporal del algoritmo original.

2.4.2. HIPÓTESIS ESPECÍFICAS

a) La complejidad temporal del algoritmo de encriptación Rijndael es O(n), la independencia matemática para los bloques encriptados es muy alta y las posibles salidas de la función ShiftRows es muy pequeña.

b) El diseño de las nuevas funciones KeyExpansion y ShiftRows aseguran una mayor entropía en las salidas.

c) La implementación de las nuevas funciones KeyExpansion y ShiftRows en el algoritmo original aseguran la entropía de las salidas y la complejidad temporal es de O(n).

d) La modificación de las funciones KeyExpansion y ShiftRows en el algoritmo Rijndael no implica un incremento en la complejidad temporal del algoritmo en general, se incrementó la entropía en los datos cifrados con la implementación de la modificación de las funcioines KeyExpansion y ShiftRows.

2.5. OPERACIONALIZACIÓN DE VARIABLES

VARIABLE	DIMENSIONES	INDICADORES
Variable independiente Modificación del algoritmo de encriptación Rijndael en el que se basa el estándar de encriptación avanzada AES	• Complejidad temporal. • Numero de posibles salidas de la función ShiftRows. • Cantidad de bits reducidos con los algoritmos de compresión.	• $O_1(n)$. • Cantidad de posibles salidas de la función a modificar. • Tamaño final del archivo encriptado vs tamaño archivo encriptado comprimido.
Variable dependiente • Analizar las funciones del algoritmo Rijndael mediante la técnica de análisis de la complejidad temporal, la entropía de sus salidas mediante la comprensión de los archivos encriptados y la cantidad de posibles salidas de la función ShiftRows.	• Complejidad temporal. • Numero de posibles salidas de la función ShiftRows. • Cantidad de bits reducidos con los algoritmos de compresión.	• $O_1(n)$. • Cantidad de posibles salidas de la función a modificar. • Tamaño final del archivo encriptado vs tamaño archivo encriptado comprimido.
• Diseñar las modificaciones para las funciones KeyExpansion y ShiftRows que aseguren una mayor aleatoriedad de los datos.	• Numero de posibles salidas de la(s) función(es) modificadas.	• Cantidad de posibles salidas de la(s) función(es) modificadas.
• Implementar las nuevas funciones KeyExpansion y ShiftRows al algoritmo Rijndael.	• Complejidad temporal. • Cantidad de bits reducidos con los algoritmos de compresión.	• $O_1(n)$ • Tamaño final del archivo encriptado vs tamaño archivo encriptado comprimido.
• Análisis y comparación de las nuevas funciones, con las funciones originales.	• Complejidad temporal del nuevo algoritmo diseñado versus el original. • Numero de posibles salidas de la(s) función(es) del algoritmo original versus número de posibles salidas de la(s) función(es) modificadas. • Cantidad de bits reducidos con los algoritmos de compresión sobre los archivos encriptados por el algoritmo original versus los generador por la modificación.	• $O_1(n)$ vs $O_2(n)$ • Cantidad de posibles salidas de la función ShiftRows() vs Cantidad de posibles salidas de la modificación de la función ShiftRows(). • Cantidad de bits reducidos con los algoritmos de compresión del algoritmo original VS cantidad de bits reducidos con los algoritmos de compresión de la modificación.

CAPÍTULO III.

DISEÑO METODOLÓGICO DE INVESTIGACIÓN

3.1. TIPO Y DISEÑO DE INVESTIGACIÓN

La presente investigación es de tipo descriptiva ya que se describen las diferentes funciones del algoritmo de encriptación AES, así como la modificación de estas y el impacto que tiene sobre el proceso de encriptación y desencriptación de datos.

Se realizó el análisis asintótico del algoritmo de encriptación AES para obtener la complejidad del algoritmo sin modificaciones para ello se consideró la notación big O u O mayúscula.

Después se procedió a modificar el algoritmo las funciones *KeyExpansion()* y *ShiftRow()* del algoritmo, teniendo las nuevas funciones se volvió a realizar el análisis asintótico considerando también la notación big O u O mayúscula, se comparó ambos resultados para medir el impacto que esta modificación tuvo en el tiempo de ejecución.

3.2. POBLACIÓN Y MUESTRA DE INVESTIGACIÓN

3.2.1. POBLACIÓN

La población está dada por todos los archivos generados manual y aleatoriamente para ser encriptados.

3.2.2. MUESTRA

La muestra está dada por todos los archivos y claves usados para la encriptación.

3.3. TÉCNICAS E INSTRUMENTOS PARA RECOLECTAR INFORMACIÓN

Para obtener la complejidad temporal de los algoritmos implementados se utilizó el análisis asintótico considerando la notación big O.

Para demostrar la independencia matemática de los archivos encriptados se usó la compresión de archivos ya que estos buscan patrones matemáticamente reducibles y de esta forma se puede conseguir una reducción de tamaño en el archivo.

Se analizó el proceso, las entradas y salidas de la función ShiftRows y la modificación de esta, para comparar la difusión de datos dada por esta función.

Se utilizó la distribución normal z para diferencia en las medias de dos muestras para comparar los diferentes tamaños de los archivos encriptados y comprimidos para medir el impacto de la modificación en la entropía ofrecida por el algoritmo original **ecuación 2:**

$$Z = \frac{(\bar{x} - \bar{y}) - (\mu_x - \mu_y)}{\sqrt{\frac{\sigma_x^2}{n_x} + \frac{\sigma_y^2}{n_y}}}$$

Sigue una distribución normal estándar

Donde:

$\bar{x} - \bar{y}$: $Diferencia\ de\ medias\ de\ las\ muestras.$

$\mu_x - \mu_y$: $Diferencias\ de\ medias\ poblacional.$

σ_x: $Desviación\ estándar\ de\ la\ primera\ muestra.$

σ_y: $Desviación\ estándar\ de\ la\ segunda\ muestra.$

n_x: $Tamaño\ de\ la\ primera\ muestra.$

n_y: $Tamaño\ de\ la\ segunda\ muestra.$

CAPÍTULO IV.

ANÁLISIS E INTERPRETACIÓN DE RESULTADOS DE LA INVESTIGACIÓN

Para cumplir con los objetivos descritos en el presente trabajo de investigación se tomó y analizo el algoritmo original con el análisis de la complejidad temporal, la entropía de las salidas del algoritmo y el número de posibles salidas de la función a modificar ShiftRows.

Se diseñó la modificación de las funciones KeyExpansion aparte de generar las sub claves necesarias para cada ronda se genera una nueva variable llamada ShiftMoves para almacenar números que determinaran los movimientos a utilizar en la función ShiftRows; en la función ShiftRows se utiliza los números almacenados en ShiftMoves para determinar los movimientos de las filas en cada ronda para cada bloque a encriptar.

Una vez diseñadas las modificaciones del algoritmo se procedió a implementar las modificaciones en el algoritmo original para medir su impacto en la complejidad temporal y la entropía de las salidas de este.

Se tomó los datos obtenidos del análisis de la complejidad temporal, cantidad posibles salidas y la entropía del algoritmo original y se la comparo con los datos obtenidos de la modificación de esta forma poder medir el impacto de la modificación.

4.1. ANÁLISIS DEL ALGORITMO ORIGINAL RIJNDAEL

Con el análisis de la complejidad temporal del algoritmo original Rijndael encontraremos una idea de cuánto tardara este en encriptar un archivo, con la

comprensión con el algoritmo de Huffman aplicada a los archivos encriptados se buscara alguna dependencia matemática entre los caracteres encriptados ya que este algoritmo para comprimir los archivos busca dependencias entre caracteres, también se analizara las posibles salidas de la función ShiftRows ya que esta junto con la función MixColumns forman parte principal para la difusión de caracteres en el algoritmo Rijndael.

4.1.1. ANÁLISIS ASINTÓTICO DEL ALGORITMO AES ORIGINAL

Se implementó el algoritmo AES, siguiendo el algoritmo publicado por el National Institute of Standards and Technology (NIST), de la siguiente forma para realizar el análisis asintótico.

Se tomó el análisis O mayúscula para el análisis asintótico del algoritmo de encriptación así como para cada una de sus funciones.

```
Cipher(){
   state = input;
   KeyExpansion(k);                          O(KeyExpansion(k))
   AddRoundKey(0);                           O(AddRoundKey(0))

   for(round=1;round<Nr;round++){            O(Nr(
       SubBytes();                             O(SubBytes())
       ShiftRows();                            O(ShiftRows())
       MixColumns();                           O(MixColumns())
       AddRoundKey(round);                     O(AddRoundKey(round))
   }                                         ))

   SubBytes();                               O(SubBytes())
   ShiftRows();                              O(ShiftRows())
   AddRoundKey(Nr);                          O(AddRoundKey(Nr))

   Out = state;

}
```

El análisis asintótico de todo el sistema de cifrado seria:

$$O(\text{KeyExpansion}) + O(AddRoundkey) + O(Nr(O(SubBytes) + O(ShiftRows) + O(MixColumns) + O(AddRoundKey))) + O(\text{subBytes}) + O(\text{ShiftRows}) + O(\text{AddRoundkey})$$

Donde:

$$Nr = número\ de\ rondas\ (10, 12\ o\ 14)$$

Por ser muy pequeño y constante el valor de Nr se desestima para el análisis asintótico por lo que la ecuación quedaría reducido a:

$$O(KeyExpansion) + O(SubBytes) + O(ShiftRow) + O(MixColumn) + O(AddRoundKey)$$

4.1.1.1. ANÁLISIS ASINTÓTICO DE LA FUNCIÓN KEYEXPANSION

Se expande la clave inicial en Nr+1 sub claves parciales que se utilizan en la ronda inicial, de donde Nr-2 son principales y la ronda final.

Siendo Nr el número de rondas.

$2b$	28	ab	09
$7e$	ae	$f7$	cf
15	$d2$	15	$4f$
16	$a6$	88	$3c$

Figura 10: Clave Inicial

Elaboración: Propia

Las palabras que ocupan la posición múltiplo de 4 $(w_4, w_8, \dots, w_{i-1})$ se calculan:

Aplicando transformaciones Rotwrod y SubBytes a la palabra anterior w_{i-1}

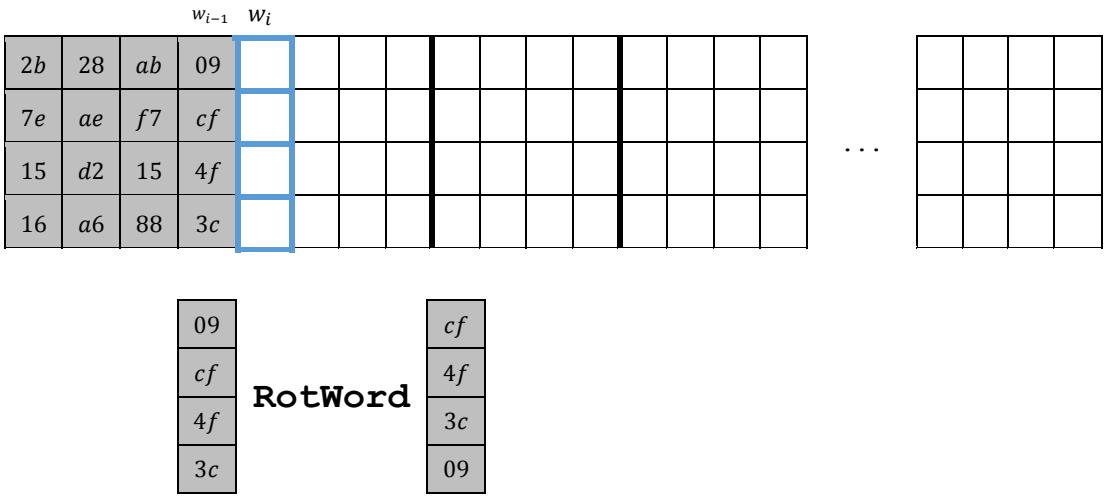
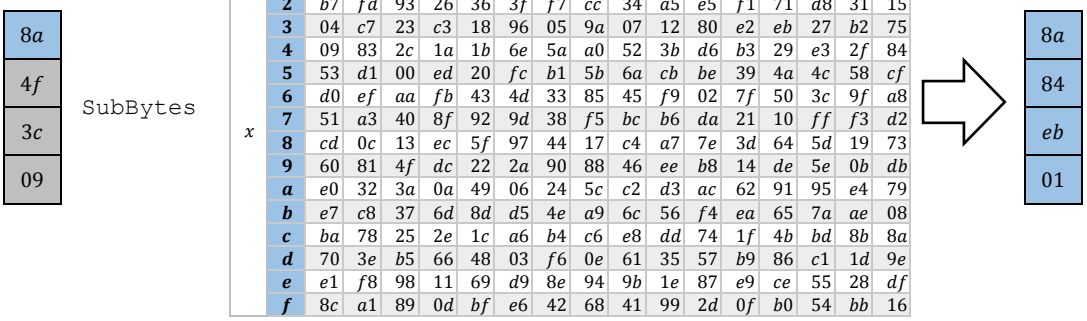


Figura 11: RotWord de KeyExpansion

Elaboración: Propia



Hex		Y															
		0	1	2	3	4	5	6	7	8	9	a	b	C	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Figura 12: SubBytes de KeyExpansion

Elaboración: Propia

Sumando (xor) el resultado obtenido en el paso anterior con la palabra de 4 posiciones antes de w_{i-4}, más una constante de ronda Rcon.

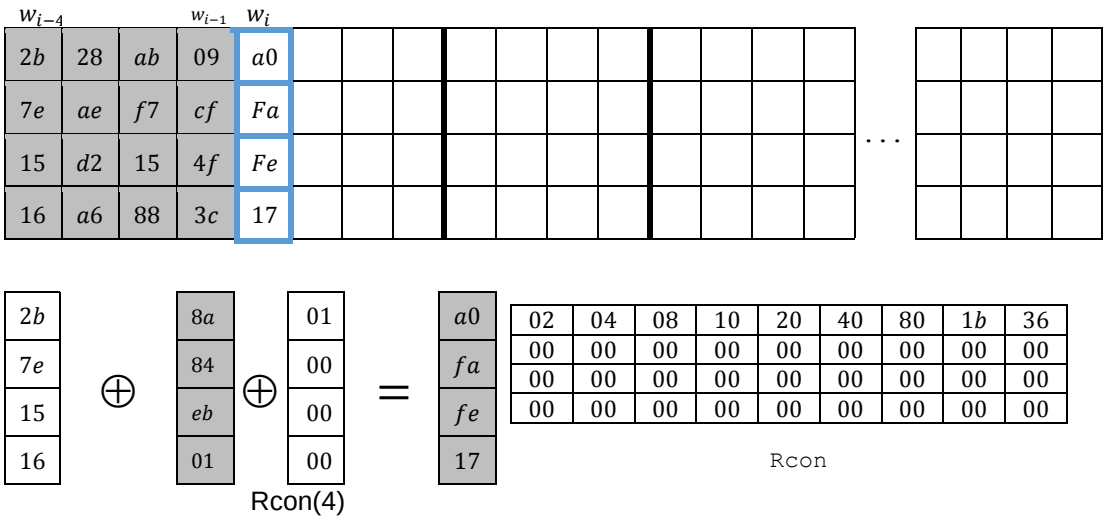


Figura 13: SubBytes de KeyExpansion

Elaboración: Propia

Las palabras restantes w_i se calculan sumando (XOR) con la palabra anterior w_{i-1} con la palabra de cuatro posiciones antes w_{i-4}.

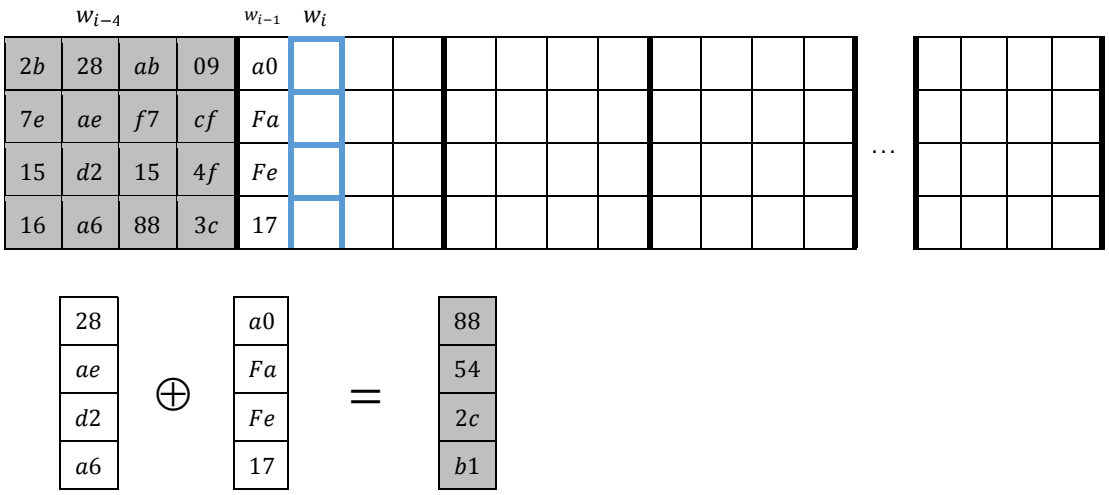


Figura 14: Cálculo de la palabra siguiente

Elaboración: Propia

De esta manera se calculan las Nr sub claves.

2b	28	ab	09	a0	88	23	2a	f2	7a	59	73	3d	47	1e	6d
7e	ae	f7	cf	fa	54	a3	6c	c2	96	35	59	80	16	23	7a
15	d2	15	4f	fe	2c	39	76	95	b9	80	f6	47	fe	7e	88
16	a6	88	3c	17	b1	39	05	f2	43	7a	7f	7d	3e	44	3b
Clave de cifrado				Sub clave de ronda 1				Sub clave de ronda 2				Sub clave de ronda 3			

d0	c9	e1	b6
14	ee	3f	63
f9	25	0c	0c
a8	89	c8	a6
Sub clave de ronda Nr			

Figura 15: Sub claves finales generadas por la funcion KeyExpansion

Elaboración: Propia

Por lo que la implementación en c++ estaría dada por:

```
KeyExpansion () {                                                    
  int i,j;                                                           
  unsigned char temp[4],k;                                           

  for (i=0; i<Nk; i++){                                              O(Nk(
    RoundKey[i*4]=Key[i*4];                                              O(1)
    RoundKey[i*4+1]=Key[i*4+1];                                          O(1)
    RoundKey[i*4+2]=Key[i*4+2];                                          O(1)
    RoundKey[i*4+3]=Key[i*4+3];}                                         O(1)))

  while (i < (Nb * (Nr+1))){                                         O(Nb * (Nr + 1)(
    for(j=0;j<4;j++){                                                    O(4)(
      temp[j]=RoundKey[(i-1) * 4 + j];                                     O(1)
    }                                                                    )
    if (i % Nk == 0){                                                O(1)
      // Función RotWord ();                                             
      {  k = temp[0];                                                O(1)
        temp[0] = temp[1];                                           O(1)
        temp[1] = temp[2];                                           O(1)
        temp[2] = temp[3];                                           O(1)
        temp[3] = k;}                                                O(1)
      // Función SubWord ();                                             
      {  temp[0]=getSBoxValue(temp[0]);                              O(getSBoxValue)
        temp[1]=getSBoxValue(temp[1]);                               O(getSBoxValue)
        temp[2]=getSBoxValue(temp[2]);                               O(getSBoxValue)
        temp[3]=getSBoxValue(temp[3]);}                              O(getSBoxValue)
      temp[0] =  temp[0] ^ Rcon[i/Nk];                               O(1)
    }
    else if (Nk > 6 && i % Nk == 4){                                 O(2)
      // Función Subword()
      {  temp[0]=getSBoxValue(temp[0]);                              O(getSBoxValue)
        temp[1]=getSBoxValue(temp[1]);                               O(getSBoxValue)
        temp[2]=getSBoxValue(temp[2]);                               O(getSBoxValue)
```

```
            temp[3]=getSBoxValue(temp[3]);}                    O(getSBoxValue)
        }
        RoundKey[i*4+0] = RoundKey[(i-Nk)*4+0] ^ temp[0];      O(1)
        RoundKey[i*4+1] = RoundKey[(i-Nk)*4+1] ^ temp[1];      O(1)
        RoundKey[i*4+2] = RoundKey[(i-Nk)*4+2] ^ temp[2];      O(1)
        RoundKey[i*4+3] = RoundKey[(i-Nk)*4+3] ^ temp[3];      O(1)
        i++;
    }
}                                                              ))
```

El análisis asintótico de la función KeyExpansion () nos da la ecuación:

$$O(Nk) + O\left(Nb\,(Nr+1)\left(O(4)*O(1) + O\left(\frac{5\ +\ 4(\text{getSBoxValue}) + 1}{Nk}\right)\right.\right.$$

$$\left.\left. + O\left(\frac{4(\text{getSBoxValue})}{Nk}\right) + O(4)\right)\right)$$

Donde:

$$Nr\ =\ Número\ de\ rondas\ (10{,}12\ o\ 14)$$

$$Nk\ =\ Número\ de\ palabras\ que\ comprenden\ la\ clave\ de\ cifrado.\ (4, 6\ o\ 8)$$

$$Nb\ =\ Numero\ de\ columnas\ siempre\ es\ 4$$

Todos los posibles valores se desestiman ya que no representan mayor complejidad ejecutarlos, por lo que la función quedaría reducida a:

$$O(k) + O\left(k\,(k+1)\left(O(4*1) + O\left(\frac{6\ +\ 4(\text{getSBoxValue})}{k}\right)\right.\right.$$

$$\left.\left. + O\left(\frac{4(\text{getSBoxValue})}{k}\right) + O(4)\right)\right)$$

Reduciendo por ambigüedad

$$O(\text{getSBoxValue})$$

La función getSBoxValue() está definida por:

```
int getSBoxValue(int num)
{
   int sbox[256] =   {
   //0    1    2    3    4    5    6    7    8    9    A    B    C    D    E    F
   0x63,0x7c,0x77,0x7b,0xf2,0x6b,0x6f,0xc5,0x30,0x01,0x67,0x2b,0xfe,0xd7,0xab,0x76,//0
   0xca,0x82,0xc9,0x7d,0xfa,0x59,0x47,0xf0,0xad,0xd4,0xa2,0xaf,0x9c,0xa4,0x72,0xc0,//1
   0xb7,0xfd,0x93,0x26,0x36,0x3f,0xf7,0xcc,0x34,0xa5,0xe5,0xf1,0x71,0xd8,0x31,0x15,//2
   0x04,0xc7,0x23,0xc3,0x18,0x96,0x05,0x9a,0x07,0x12,0x80,0xe2,0xeb,0x27,0xb2,0x75,//3
   0x09,0x83,0x2c,0x1a,0x1b,0x6e,0x5a,0xa0,0x52,0x3b,0xd6,0xb3,0x29,0xe3,0x2f,0x84,//4
   0x53,0xd1,0x00,0xed,0x20,0xfc,0xb1,0x5b,0x6a,0xcb,0xbe,0x39,0x4a,0x4c,0x58,0xcf,//5
   0xd0,0xef,0xaa,0xfb,0x43,0x4d,0x33,0x85,0x45,0xf9,0x02,0x7f,0x50,0x3c,0x9f,0xa8,//6
   0x51,0xa3,0x40,0x8f,0x92,0x9d,0x38,0xf5,0xbc,0xb6,0xda,0x21,0x10,0xff,0xf3,0xd2,//7
   0xcd,0x0c,0x13,0xec,0x5f,0x97,0x44,0x17,0xc4,0xa7,0x7e,0x3d,0x64,0x5d,0x19,0x73,//8
   0x60,0x81,0x4f,0xdc,0x22,0x2a,0x90,0x88,0x46,0xee,0xb8,0x14,0xde,0x5e,0x0b,0xdb,//9
   0xe0,0x32,0x3a,0x0a,0x49,0x06,0x24,0x5c,0xc2,0xd3,0xac,0x62,0x91,0x95,0xe4,0x79,//A
   0xe7,0xc8,0x37,0x6d,0x8d,0xd5,0x4e,0xa9,0x6c,0x56,0xf4,0xea,0x65,0x7a,0xae,0x08,//B
   0xba,0x78,0x25,0x2e,0x1c,0xa6,0xb4,0xc6,0xe8,0xdd,0x74,0x1f,0x4b,0xbd,0x8b,0x8a,//C
   0x70,0x3e,0xb5,0x66,0x48,0x03,0xf6,0x0e,0x61,0x35,0x57,0xb9,0x86,0xc1,0x1d,0x9e,//D
   0xe1,0xf8,0x98,0x11,0x69,0xd9,0x8e,0x94,0x9b,0x1e,0x87,0xe9,0xce,0x55,0x28,0xdf,//E
   0x8c,0xa1,0x89,0x0d,0xbf,0xe6,0x42,0x68,0x41,0x99,0x2d,0x0f,0xb0,0x54,0xbb,0x16 //F
   };                                                                              O(1)
   return sbox[num];                                                               O(1)
}
```

La ecuación para definir getSboxValue estaría dada por

$$O(1) + O(1)$$

La podemos reducir a:

$$O(k)$$

Reemplazando en la ecuación de la función KeyExpansion

$$\mathrm{O}(\mathrm{getSBoxValue})$$

$$O\big(O(k)\big)$$

Podemos asumir la complejidad temporal de la función KeyExpansion:

$$O(k)$$

4.1.1.2. ANÁLISIS ASINTÓTICO DE LA FUNCIÓN SUBBYTES

Esta función se basa en sustitución de bytes con la tabla S-box

19	$a0$	$9a$	$e9$
$3d$	$f4$	$c6$	$f8$
$e3$	$e2$	$8d$	48
be	$2b$	$2a$	08

Figura 16: Estado inicial para Subbytes

Elaboración: Propia

La sustitución SubBytes para "**19**"

Hex		Y															
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	***a***	***b***	***c***	***d***	***e***	***f***
X	**0**	63	$7c$	77	$7b$	$f2$	$6b$	$6f$	$c5$	30	01	67	$2b$	fe	$d7$	ab	76
	1	Ca	82	$c9$	$7d$	fa	59	47	$f0$	ad	$d4$	$a2$	af	$9c$	$a4$	72	$c0$
	2	$b7$	fd	93	26	36	$3f$	$f7$	cc	34	$a5$	$e5$	$f1$	71	$d8$	31	15
	3	04	$c7$	23	$c3$	18	96	05	$9a$	07	12	80	$e2$	eb	27	$b2$	75
	4	09	83	$2c$	$1a$	$1b$	$6e$	$5a$	$a0$	52	$3b$	$d6$	$b3$	29	$e3$	$2f$	84
	5	53	$d1$	00	Ed	20	fc	$b1$	$5b$	$6a$	cb	be	39	$4a$	$4c$	58	cf
	6	$d0$	ef	aa	fb	43	$4d$	33	85	45	$f9$	02	$7f$	50	$3c$	$9f$	$a8$
	7	51	$a3$	40	$8f$	92	$9d$	38	$f5$	bc	$b6$	da	21	10	ff	$f3$	$d2$
	8	cd	$0c$	13	ec	$5f$	97	44	17	$c4$	$a7$	$7e$	$3d$	64	$5d$	19	73
	9	60	81	$4f$	dc	22	$2a$	90	88	46	ee	$b8$	14	de	$5e$	$0b$	db
	a	$e0$	32	$3a$	$0a$	49	06	24	$5c$	$c2$	$d3$	ac	62	91	95	$e4$	79
	b	$e7$	$c8$	37	$6d$	$8d$	$d5$	$4e$	$a9$	$6c$	56	$f4$	ea	65	$7a$	ae	08
	c	ba	78	25	$2e$	$1c$	$a6$	$b4$	$c6$	$e8$	dd	74	$1f$	$4b$	bd	$8b$	$8a$
	d	70	$3e$	$b5$	66	48	03	$f6$	$0e$	61	35	57	$b9$	86	$c1$	$1d$	$9e$
	e	$e1$	$f8$	98	11	69	$d9$	$8e$	94	$9b$	$1e$	87	$e9$	ce	55	28	df
	f	$8c$	$a1$	89	$0d$	bf	$e6$	42	68	41	99	$2d$	$0f$	$b0$	54	bb	16

Figura 17: Tabla S-box

Elaboración: Propia

Se remplaza el valor encontrado en S-box para "**19**"

$d4$	$a0$	$9a$	$e9$
$3d$	$f4$	$c6$	$f8$
$e3$	$e2$	$8d$	48
be	$2b$	$2a$	08

Figura 18: Remplazo en SubBytes

Elaboración: Propia

Se continúa con los remplazos hasta terminar con todo el bloque, el estado final de este estaría dado por:

$d4$	$e0$	$b8$	$1e$
27	bf	$b4$	41
11	98	$5d$	52
ae	$f1$	$e3$	30

Figura 19: Estado final SubBytes

Elaboración: Propia

La implementación del algoritmo en c++ es:

```
SubBytes(){
   int i,j;
   for(i=0;i<4;i++){
      for(j=0;j<4;j++){
       state[i][j] = getSBoxValue(state[i][j]);
      }
   }
}
```

```
O(4(
 O(4(
  O(getSBoxValue)
 ))
))
```

El análisis asintótico de la función SubBytes () nos da la ecuación:

$$O\big(4 * O\big(4 * O(getSBoxValue)\big)\big)$$

En la función KeyExpansion analizamos la función getSBoxValue obteniéndose de la complejidad temporal $O(k)$

Por lo que la ecuación para la función SubBytes quedaría como:

$$O\left(4 * O\left(4 * O(k)\right)\right)$$

Resolviendo esta ecuación se obtiene que la complejidad temporal de la función SubBytes es:

$$O(k)$$

Donde k es una constante

4.1.1.3. ANÁLISIS ASINTÓTICO DE LA FUNCIÓN SHIFTROWS

En la función ShiftRows cada byte de la segunda fila se desplaza una posición a la izquierda; la tercera fila dos posiciones y la cuarta tres.

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$

ShiftRows ()
→

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$
$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,0}$
$a_{2,2}$	$a_{2,3}$	$a_{2,0}$	$a_{2,1}$
$a_{3,3}$	$a_{3,0}$	$a_{3,1}$	$a_{3,2}$

Figura 20: Desplazamiento de la función ShiftRows

Elaboración: Propia

Por lo que la implementación de la función ShiftRows se haría de la siguiente manera:

```
ShiftRows(){
  unsigned char temp;

  // Rotar la segunda fila 1 pos a la izquierda
  temp=state[1][0];                                   O(1)
  state[1][0]=state[1][1];                            O(1)
  state[1][1]=state[1][2];                            O(1)
  state[1][2]=state[1][3];                            O(1)
  state[1][3]=temp;                                   O(1)

  // Rotar la tercera fila 2 pos a la izquierda
  temp=state[2][0];                                   O(1)
  state[2][0]=state[2][2];                            O(1)
  state[2][2]=temp;                                   O(1)
  temp=state[2][1];                                   O(1)
  state[2][1]=state[2][3];                            O(1)
  state[2][3]=temp;                                   O(1)

  // Rotar la cuarta fila 1 pos a la izquierda
  temp=state[3][0];                                   O(1)
  state[3][0]=state[3][3];                            O(1)
  state[3][3]=state[3][2];                            O(1)
  state[3][2]=state[3][1];                            O(1)
  state[3][1]=temp;                                   O(1)
}
```

La ecuación para determinar la complejidad temporal de la función ShiftRows es:

$$O(5) + O(6) + O(5)$$

La cual se puede reducir a:

$$O(k)$$

Donde k es el número de asignaciones realizadas (16).

4.1.1.4. ANÁLISIS ASINTÓTICO DE LA FUNCIÓN MIXCOLUMNS

Los cuatro bytes de cada columna son multiplicados dentro del campo de galois por una matriz definida.

$d4$	$e0$	$b8$	$1e$
bf	$b4$	41	27
$5d$	52	11	98
30	ae	$f1$	$e3$

Figura 21: Estado inicial para Mixcolumns

Elaboración: Propia

La multiplicación por la matriz dada dentro del Campo de Galois estaría dada por:

$$\begin{bmatrix} d4 \\ bf \\ 5d \\ 30 \end{bmatrix} \cdot \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} = \begin{bmatrix} 04 \\ 66 \\ 81 \\ e5 \end{bmatrix}$$

Obtenido este resultado, se reemplaza en la matriz original.

04	$e0$	$b8$	$1e$
66	$b4$	41	27
81	52	11	98
$e5$	ae	$f1$	$e3$

Figura 22: Primer reemplazo de la función MixColumns

Elaboración: Propia

Se procede de la misma manera hasta recorrer toda la matriz de estado

04	$e0$	48	28
66	bc	$f8$	06
81	19	$d3$	26
$e5$	$9a$	$7a$	$4c$

Figura 23 Matriz final de la función MixColumns

Elaboración: Propia

De esta forma la implementación de la function MixColumns estaría dada por:

```
MixColumns(){                                                          O(4)(
  int i;
  unsigned char Tmp,Tm,t;
  for (I = 0; i < 4; i++){
    t=state[0][i];                                                     O(1)

    Tmp = state[0][i]^state[1][i]^state[2][i]^state[3][i];              O(1)
    Tm = state[0][i] ^ state[1][i];                                    O(1)
    Tm = xtime(Tm);                                                    O(1)

    state[0][i] ^= Tm ^ Tmp;                                           O(1)
    Tm = state[1][i] ^ state[2][i];                                    O(1)
    Tm = xtime(Tm);                                                    O(1)

    state[1][i] ^= Tm ^ Tmp;                                           O(1)
    Tm = state[2][i] ^ state[3][i];                                    O(1)
    Tm = xtime(Tm); state[2][i] ^= Tm ^ Tmp;                           O(1)
    Tm = state[3][i] ^ t;                                              O(1)
    Tm = xtime(Tm);                                                    O(1)
    state[3][i] ^= Tm ^ Tmp;                                           O(1)
  }                                                                    )
}
```

El análisis asintótico de la función MixColumns () nos da la ecuación:

$$O(4 * O(13)) = O(k)$$

Donde k es un valor muy pequeño.

4.1.1.5. ANÁLISIS ASINTÓTICO DE LA FUNCIÓN ADDROUNDKEY

Esta función la suma (XOR) la matriz de estado con cada subclave calculada en la función KeyExpansion para cada ronda.

04	$e0$	48	28
66	bc	$f8$	06
81	19	$d3$	26
$e5$	$9a$	$7a$	$4c$

$a0$	88	23	$2a$
fa	54	$a3$	$6c$
fe	$2c$	39	76
17	$b1$	39	05

Figura 24: Estado inicial y sub clave inicial para la función AddRoundKey

Fuente: Elaboración propia

Sumando (XOR) la primera columna de estado con la primera columna de la subclave calculada en KeyExpansion.

$$\begin{bmatrix}04\\66\\81\\e5\end{bmatrix} \oplus \begin{bmatrix}48\\f8\\d3\\7a\end{bmatrix} = \begin{bmatrix}28\\06\\26\\4c\end{bmatrix}$$

Figura 25: Suma (XOR) de la función AddRoundKey

Elaboración: Propia

La implementación para la función AddRoundKey estaría dada por:

```
AddRoundKey(int round){
 int i, j;

 for(i = 0; i < 4; i++){                                    O(4(
  for(j = 0; j < 4; j++){                                    O(4(
   state[j][i]^=RoundKey[round*Nb*4+i*Nb+j];                   O(1)
  }                                                           ))
 }                                                           ))
}
```

La ecuación para calcular la complejidad temporal de la función AddRoundKey está dada por:

$$O\left(4\left(O(4(1))\right)\right)$$

Y esta se puede reducir a:

$$O(k)$$

Reemplazando en la ecuación general del análisis asintótico de todo el algoritmo tenemos

$$\mathrm{O}(KeyExpansion) + \mathrm{O}(SubBytes) + \mathrm{O}(ShiftRow) + \mathrm{O}(MixColumn)$$

$$+ \mathrm{O}(AddRoundKey) = \mathrm{O}(k) + \mathrm{O}(\mathrm{k}) + \mathrm{O}(k) + \mathrm{O}(k) + \mathrm{O}(k)$$

Reduciendo la ecuación se tiene

$$\mathrm{O}(k)$$

Es decir la complejidad computacional del AES siempre tiende a $O(k)$ donde k depende del tamaño de la clave, con los reemplazos con la tabla sBox, las operaciones xor la tabla Rcon, la rotación de las funciones ShiftRow y la multiplicación con la tabla xTime de MixColumns.

Debemos tener en cuenta que AES se trata de un cifrado por bloques, es decir, primero cifra un pedazo del texto en plano, para después continuar con otro pedazo hasta que el texto este completamente cifrado, por lo que la complejidad depende del tamaño del texto multiplicado por la complejidad del algoritmo, por lo que tenemos que la complejidad final está dada por:

$$O(n) * O(k) = O(n * k)$$

Donde N es el tamaño del bloque a cifrar y k es la complejidad constante del algoritmo AES.

4.1.2. ENTROPÍA DE LAS SALIDAS DEL ALGORITMO AES ORIGINAL

En esta parte de la investigación se implementó el código en c++ siguiendo el algoritmo publicado por la FIPS, después de realizada la implementación se procedió a encriptar archivos de texto de diferentes tamaños, generados con caracteres al azar por otro programa.

Después se procedió a compartir los archivos encriptados ya que la compresión de datos se basa en buscar patrones o dependencias matemáticas y resumirlos, es decir si se quiere comprimir un archivo que contiene la serie “AAAAAA” que sería el equivalente a 6 bytes se puede almacenar mejor en “6A” que literalmente ocuparía 2 bytes; utilizando algoritmos de compresión sobre los

archivos encriptados se puede demostrar la independencia matemática o aleatoriedad de sus caracteres.

Para analizar la entropía del algoritmo original se realizaron 2 tipos de experimentos uno para el mejor caso, en el que los caracteres son repetitivos y otro para el peor caso en el que todos los caracteres son aleatorios, utilizando los algoritmos deflate y LZMA.

4.1.2.1. PEOR CASO

Para las pruebas con el peor caso para ambos algoritmos se consideró una clave fuerte como **1Ds$f23-USAbCc12**, y archivos con caracteres aleatorios de diferentes tamaño, para de esta forma crear las condiciones óptimas para observar los efectos del algoritmo en archivos aleatorios.

Se utilizó la siguiente configuración con el programa 7-zip y diccionarios para los diferentes tamaños de archivo, y diferentes tipos de compresión tanto para los archivos encriptados generados, tanto para la primera clave como para la segunda clave.

Formato	Zip	**Tipo de compresión**	LZMA
Nivel de compresión	Ultra	**Tamaño de palabra**	8
Numero de hilos de la CPU	8		

Archivos	**Tamaño**	**Diccionario**
output1.txt	762 MB	256 MB
output2.txt – output11.txt	625 KB	1 MB
output12.txt – output15.txt	1.98 MB	2 MB
output16.txt – output25.txt	3.05 MB	3 MB

output26.txt – output30.txt	305 MB	256 MB
output31.txt – output40.txt	457 MB	256 MB
output41.txt – output45.txt	7.81 KB	64 KB
output46.txt – output50.txt	76.2 MB	96 MB

Tabla 32: Configuración LZMA utilizada para comprimir los archivos generados para el peor caso con el algoritmo original.

Elaboración: Propia

Formato	Zip	**Tipo de compresión**	Deflate
Nivel de compresión	Ultra	**Tamaño de palabra**	8
Numero de hilos de la CPU	8	**Tamaño de diccionario**	32 KB

Tabla 33: Configuración Deflate utilizada para comprimir los archivos generados para el peor caso con el algoritmo original.

Elaboración: Propia

Clave: 1Ds$f23-USAbCc12			
Nombre del archivo	**Tamaño del archivo**	**Tamaño de archivo comprimido (LZMA)**	**Tamaño de archivo comprimido (Deflate)**
output1.txt	762 MB (800,000,000 bytes)	762 MB (800,000,156 bytes)	762 MB (800,000,156 bytes)
output2.txt	625 KB (640,000 bytes)	625 KB (640,156 bytes)	625 KB (640,156 bytes)
output3.txt	625 KB (640,000 bytes)	625 KB (640,156 bytes)	625 KB (640,156 bytes)
output4.txt	625 KB (640,000 bytes)	625 KB (640,156 bytes)	625 KB (640,156 bytes)

output5.txt	625 KB (640,000 bytes)	625 KB (640,156 bytes)	625 KB (640,156 bytes)
output6.txt	625 KB (640,000 bytes)	625 KB (640,156 bytes)	625 KB (640,156 bytes)
output7.txt	625 KB (640,000 bytes)	625 KB (640,156 bytes)	625 KB (640,156 bytes)
output8.txt	625 KB (640,000 bytes)	625 KB (640,156 bytes)	625 KB (640,156 bytes)
output9.txt	625 KB (640,000 bytes)	625 KB (640,156 bytes)	625 KB (640,156 bytes)
output10.txt	625 KB (640,000 bytes)	625 KB (640,158 bytes)	625 KB (640,158 bytes)
output11.txt	625 KB (640,000 bytes)	625 KB (640,158 bytes)	625 KB (640,158 bytes)
output12.txt	1.98 MB (2,080,000 bytes)	1.98 MB (2,080,158 bytes)	1.98 MB (2,080,158 bytes)
output13.txt	1.98 MB (2,080,000 bytes)	1.98 MB (2,080,158 bytes)	1.98 MB (2,080,158 bytes)
output14.txt	1.98 MB (2,080,000 bytes)	1.98 MB (2,080,158 bytes)	1.98 MB (2,080,158 bytes)
output15.txt	1.98 MB (2,080,000 bytes)	1.98 MB (2,080,158 bytes)	1.98 MB (2,080,158 bytes)
output16.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,158 bytes)	3.05 MB (3,200,158 bytes)
output17.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,158 bytes)	3.05 MB (3,200,158 bytes)
output18.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,158 bytes)	3.05 MB (3,200,158 bytes)
output19.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,158 bytes)	3.05 MB (3,200,158 bytes)
output20.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,158 bytes)	3.05 MB (3,200,158 bytes)
output21.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,158 bytes)	3.05 MB (3,200,158 bytes)
output22.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,158 bytes)	3.05 MB (3,200,158 bytes)

output23.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,158 bytes)	3.05 MB (3,200,158 bytes)
output24.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,158 bytes)	3.05 MB (3,200,158 bytes)
output25.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,158 bytes)	3.05 MB (3,200,158 bytes)
output26.txt	305 MB (320,000,000 bytes)	305 MB (320,000,158 bytes)	305 MB (320,000,158 bytes)
output27.txt	305 MB (320,000,000 bytes)	305 MB (320,000,158 bytes)	305 MB (320,000,158 bytes)
output28.txt	305 MB (320,000,000 bytes)	305 MB (320,000,158 bytes)	305 MB (320,000,158 bytes)
output29.txt	305 MB (320,000,000 bytes)	305 MB (320,000,158 bytes)	305 MB (320,000,158 bytes)
output30.txt	305 MB (320,000,000 bytes)	305 MB (320,000,158 bytes)	305 MB (320,000,158 bytes)
output31.txt	457 MB (480,000,000 bytes)	457 MB (480,000,158 bytes)	457 MB (480,000,158 bytes)
output32.txt	457 MB (480,000,000 bytes)	457 MB (480,000,158 bytes)	457 MB (480,000,158 bytes)
output33.txt	457 MB (480,000,000 bytes)	457 MB (480,000,158 bytes)	457 MB (480,000,158 bytes)
output34.txt	457 MB (480,000,000 bytes)	457 MB (480,000,158 bytes)	457 MB (480,000,158 bytes)
output35.txt	457 MB (480,000,000 bytes)	457 MB (480,000,158 bytes)	457 MB (480,000,158 bytes)
output36.txt	457 MB (480,000,000 bytes)	457 MB (480,000,158 bytes)	457 MB (480,000,158 bytes)
output37.txt	457 MB (480,000,000 bytes)	457 MB (480,000,158 bytes)	457 MB (480,000,158 bytes)
output38.txt	457 MB (480,000,000 bytes)	457 MB (480,000,158 bytes)	457 MB (480,000,158 bytes)
output39.txt	457 MB (480,000,000 bytes)	457 MB (480,000,158 bytes)	457 MB (480,000,158 bytes)
output40.txt	457 MB (480,000,000 bytes)	457 MB (480,000,158 bytes)	457 MB (480,000,158 bytes)

output41.txt	7.81 KB (8,000 bytes)	7.96 KB (8,158 bytes)	7.96 KB (8,158 bytes)
output42.txt	7.81 KB (8,000 bytes)	7.96 KB (8,158 bytes)	7.96 KB (8,158 bytes)
output43.txt	7.81 KB (8,000 bytes)	7.96 KB (8,158 bytes)	7.96 KB (8,158 bytes)
output44.txt	7.81 KB (8,000 bytes)	7.96 KB (8,158 bytes)	7.96 KB (8,158 bytes)
output45.txt	7.81 KB (8,000 bytes)	7.96 KB (8,158 bytes)	7.96 KB (8,158 bytes)
output46.txt	76.2 MB (80,000,000 bytes)	76.2 MB (80,000,158 bytes)	76.2 MB (80,000,158 bytes)
output47.txt	76.2 MB (80,000,000 bytes)	76.2 MB (80,000,158 bytes)	76.2 MB (80,000,158 bytes)
output48.txt	76.2 MB (80,000,000 bytes)	76.2 MB (80,000,158 bytes)	76.2 MB (80,000,158 bytes)
output49.txt	76.2 MB (80,000,000 bytes)	76.2 MB (80,000,158 bytes)	76.2 MB (80,000,158 bytes)
output50.txt	76.2 MB (80,000,000 bytes)	76.2 MB (80,000,158 bytes)	76.2 MB (80,000,158 bytes)

Tabla 34: Archivos encriptados con el algoritmo original AES comprimidos con la clave 1Ds$f23-USAbCc12 (peor caso)

Elaboración: Propia

Clave: abcdefghijklmnop			
Nombre del archivo	**Tamaño del archivo**	**Tamaño del archivo comprimido (LZMA)**	**Tamaño del archivo comprimido (deflate)**
output1b.txt	762 MB (800,000,000 bytes)	762 MB (800,000,158 bytes)	762 MB (800,000,158 bytes)
output2b.txt	625 KB (640,000 bytes)	625 KB (640,158 bytes)	625 KB (640,158 bytes)

output3b.txt	625 KB (640,000 bytes)	625 KB (640,158 bytes)	625 KB (640,158 bytes)
output4b.txt	625 KB (640,000 bytes)	625 KB (640,158 bytes)	625 KB (640,158 bytes)
output5b.txt	625 KB (640,000 bytes)	625 KB (640,158 bytes)	625 KB (640,158 bytes)
output6b.txt	625 KB (640,000 bytes)	625 KB (640,158 bytes)	625 KB (640,158 bytes)
output7b.txt	625 KB (640,000 bytes)	625 KB (640,158 bytes)	625 KB (640,158 bytes)
output8b.txt	625 KB (640,000 bytes)	625 KB (640,158 bytes)	625 KB (640,158 bytes)
output9b.txt	625 KB (640,000 bytes)	625 KB (640,158 bytes)	625 KB (640,158 bytes)
output10b.txt	625 KB (640,000 bytes)	625 KB (640,158 bytes)	625 KB (640,160 bytes)
output11b.txt	625 KB (640,000 bytes)	625 KB (640,158 bytes)	625 KB (640,160 bytes)
output12b.txt	1.98 MB (2,080,000 bytes)	1.98 MB (2,080,160 bytes)	1.98 MB (2,080,160 bytes)
output13b.txt	1.98 MB (2,080,000 bytes)	1.98 MB (2,080,160 bytes)	1.98 MB (2,080,160 bytes)
output14b.txt	1.98 MB (2,080,000 bytes)	1.98 MB (2,080,160 bytes)	1.98 MB (2,080,160 bytes)
output15b.txt	1.98 MB (2,080,000 bytes)	1.98 MB (2,080,160 bytes)	1.98 MB (2,080,160 bytes)
output16b.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,160 bytes)	3.05 MB (3,200,160 bytes)
output17b.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,160 bytes)	3.05 MB (3,200,160 bytes)
output18b.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,160 bytes)	3.05 MB (3,200,160 bytes)
output19b.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,160 bytes)	3.05 MB (3,200,160 bytes)
output20b.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,160 bytes)	3.05 MB (3,200,160 bytes)
output21b.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,160 bytes)	3.05 MB (3,200,160 bytes)
output22b.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,160 bytes)	3.05 MB (3,200,160 bytes)

output23b.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,160 bytes)	3.05 MB (3,200,160 bytes)
output24b.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,160 bytes)	3.05 MB (3,200,160 bytes)
output25b.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,160 bytes)	3.05 MB (3,200,160 bytes)
output26b.txt	305 MB (320,000,000 bytes)	305 MB (320,000,160 bytes)	305 MB (320,000,160 bytes)
output27b.txt	305 MB (320,000,000 bytes)	305 MB (320,000,160 bytes)	305 MB (320,000,160 bytes)
output28b.txt	305 MB (320,000,000 bytes)	305 MB (320,000,160 bytes)	305 MB (320,000,160 bytes)
output29b.txt	305 MB (320,000,000 bytes)	305 MB (320,000,160 bytes)	305 MB (320,000,160 bytes)
output30b.txt	305 MB (320,000,000 bytes)	305 MB (320,000,160 bytes)	305 MB (320,000,160 bytes)
output31b.txt	457 MB (480,000,000 bytes)	457 MB (480,000,160 bytes)	457 MB (480,000,160 bytes)
output32b.txt	457 MB (480,000,000 bytes)	457 MB (480,000,160 bytes)	457 MB (480,000,160 bytes)
output33b.txt	457 MB (480,000,000 bytes)	457 MB (480,000,160 bytes)	457 MB (480,000,160 bytes)
output34b.txt	457 MB (480,000,000 bytes)	457 MB (480,000,160 bytes)	457 MB (480,000,160 bytes)
output35b.txt	457 MB (480,000,000 bytes)	457 MB (480,000,160 bytes)	457 MB (480,000,160 bytes)
output36b.txt	457 MB (480,000,000 bytes)	457 MB (480,000,160 bytes)	457 MB (480,000,160 bytes)
output37b.txt	457 MB (480,000,000 bytes)	457 MB (480,000,160 bytes)	457 MB (480,000,160 bytes)
output38b.txt	457 MB (480,000,000 bytes)	457 MB (480,000,160 bytes)	457 MB (480,000,160 bytes)
output39b.txt	457 MB (480,000,000 bytes)	457 MB (480,000,160 bytes)	457 MB (480,000,160 bytes)
output40b.txt	457 MB (480,000,000 bytes)	457 MB (480,000,160 bytes)	457 MB (480,000,160 bytes)
output41b.txt	7.81 KB (8,000 bytes)	7.96 KB (8,160 bytes)	7.96 KB (8,160 bytes)
output42b.txt	7.81 KB (8,000 bytes)	7.96 KB (8,160 bytes)	7.96 KB (8,160 bytes)

output43b.txt	7.81 KB (8,000 bytes)	7.96 KB (8,160 bytes)	7.96 KB (8,160 bytes)
output44b.txt	7.81 KB (8,000 bytes)	7.96 KB (8,160 bytes)	7.96 KB (8,160 bytes)
output45b.txt	7.81 KB (8,000 bytes)	7.96 KB (8,160 bytes)	7.96 KB (8,160 bytes)
output46b.txt	76.2 MB (80,000,000 bytes)	76.2 MB (80,000,160 bytes)	76.2 MB (80,000,160 bytes)
output47b.txt	76.2 MB (80,000,000 bytes)	76.2 MB (80,000,160 bytes)	76.2 MB (80,000,160 bytes)
output48b.txt	76.2 MB (80,000,000 bytes)	76.2 MB (80,000,160 bytes)	76.2 MB (80,000,160 bytes)
output49b.txt	76.2 MB (80,000,000 bytes)	76.2 MB (80,000,160 bytes)	76.2 MB (80,000,160 bytes)
output50b.txt	76.2 MB (80,000,000 bytes)	76.2 MB (80,000,160 bytes)	76.2 MB (80,000,160 bytes)

Tabla 35: Archivos encriptados con el algoritmo original AES comprimidos con la clave abcdefghijklmnop (peor caso)

Elaboración: Propia

Al comprimir los archivos generados por el algoritmo de encriptación estos aumentan de tamaño ya que estos archivos aseguran una alta entropía entre los caracteres, y se agrega los headers o cabeceras propios del algoritmo de compresión.

4.1.2.2. MEJOR CASO

Para el mejor caso se tomó la clave **aaaaaaaaaaaaaaaa** y se generó archivos de un solo carácter repetido varias veces similares a “**rrrrrrrrrrrrrrr**”.

Se utilizó la siguiente configuración con el programa 7-zip y diccionarios para los diferentes tamaños de archivo, tanto para los archivos generados con la clave 1 como para la clave 2.

Formato	Zip	**Tipo de compresión**	LZMA
Nivel de compresión	Ultra	**Tamaño de palabra**	8
Numero de hilos de la CPU	8		

Archivos	**Tamaño**	**Diccionario**
output1s.txt	38.1 MB	48 MB
output2s.txt – output11s.txt	78.1 KB	64 KB
output12.txt – output21.txt	781 KB	1 MB
output22.txt – output31.txt	7.81 KB	64 KB
output32.txt – output40.txt	6.25 KB	64 KB
output41.txt	78.1 KB	64 KB
output42.txt – output45.txt	7.81 KB	64 KB
output46.txt	76.2 MB	96 MB
output47.txt – output50.txt	78.1 KB	64 KB

Tabla 36: Configuración LZMA utilizada para comprimir los archivos generados para el mejor caso con el algoritmo original.

Elaboración: Propia

Formato	Zip	**Tipo de compresión**	Deflate
Nivel de compresión	Ultra	**Tamaño de palabra**	8
Numero de hilos de la CPU	8	**Tamaño de diccionario**	32 KB

Tabla 37: Configuración Deflate utilizada para comprimir los archivos generados para el mejor caso con el algoritmo original.

Elaboración: Propia

Clave: aaaaaaaaaaaaaaaa			
Nombre del archivo	**Tamaño del archivo**	**Tamaño de archivo comprimido (LZMA)**	**Tamaño de archivo comprimido (deflate)**
output1s.txt	38.1 MB (40,000,000 bytes)	5.76 KB (5,899 bytes)	84.5 KB (86,549 bytes)
output2s.txt	78.1 KB (80,000 bytes)	265 bytes (265 bytes)	371 bytes (371 bytes)
output3s.txt	78.1 KB (80,000 bytes)	265 bytes (265 bytes)	367 bytes (367 bytes)
output4s.txt	7.81 KB (8,000 bytes)	265 bytes (265 bytes)	369 bytes (369 bytes)
output5s.txt	78.1 KB (80,000 bytes)	265 bytes (265 bytes)	370 bytes (370 bytes)
output6s.txt	78.1 KB (80,000 bytes)	265 bytes (265 bytes)	372 bytes (372 bytes)
output7s.txt	78.1 KB (80,000 bytes)	265 bytes (265 bytes)	368 bytes (368 bytes)
output8s.txt	78.1 KB (80,000 bytes)	265 bytes (265 bytes)	370 bytes (370 bytes)
output9s.txt	78.1 KB (80,000 bytes)	265 bytes (265 bytes)	367 bytes (367 bytes)
output10s.txt	78.1 KB (80,000 bytes)	265 bytes (265 bytes)	370 bytes (370 bytes)
output11s.txt	78.1 KB (80,000 bytes)	267 bytes (267 bytes)	372 bytes (372 bytes)
output12s.txt	781 KB (800,000 bytes)	370 bytes (370 bytes)	1.88 KB (1,928 bytes)
output13s.txt	781 KB (800,000 bytes)	371 bytes (371 bytes)	1.88 KB (1,930 bytes)
output14s.txt	781 KB (800,000 bytes)	370 bytes (370 bytes)	1.88 KB (1,928 bytes)
output15s.txt	781 KB (800,000 bytes)	371 bytes (371 bytes)	1.88 KB (1,930 bytes)
output16s.txt	781 KB (800,000 bytes)	371 bytes (371 bytes)	1.88 KB (1,928 bytes)

output17s.txt	781 KB (800,000 bytes)	371 bytes (371 bytes)	1.88 KB (1,930 bytes)
output18s.txt	781 KB (800,000 bytes)	371 bytes (371 bytes)	1.88 KB (1,929 bytes)
output19s.txt	781 KB (800,000 bytes)	371 bytes (371 bytes)	1.88 KB (1,927 bytes)
output20s.txt	781 KB (800,000 bytes)	371 bytes (371 bytes)	1.88 KB (1,929 bytes)
output21s.txt	781 KB (800,000 bytes)	371 bytes (371 bytes)	1.88 KB (1,927 bytes)
output22s.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	219 bytes (219 bytes)
output23s.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	219 bytes (219 bytes)
output24s.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	217 bytes (217 bytes)
output25s.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	220 bytes (220 bytes)
output26s.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	220 bytes (220 bytes)
output27s.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	218 bytes (218 bytes)
output28s.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	220 bytes (220 bytes)
output29s.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	220 bytes (220 bytes)
output30s.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	219 bytes (219 bytes)
output31s.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	219 bytes (219 bytes)
output32s.txt	6.25 KB (6,400 bytes)	223 bytes (223 bytes)	218 bytes (218 bytes)
output33s.txt	6.25 KB (6,400 bytes)	223 bytes (223 bytes)	217 bytes (217 bytes)
output34s.txt	6.25 KB (6,400 bytes)	223 bytes (223 bytes)	217 bytes (217 bytes)

output35s.txt	6.25 KB (6,400 bytes)	223 bytes (223 bytes)	215 bytes (215 bytes)
output36s.txt	6.25 KB (6,400 bytes)	223 bytes (223 bytes)	216 bytes (216 bytes)
output37s.txt	6.25 KB (6,400 bytes)	223 bytes (223 bytes)	217 bytes (217 bytes)
output38s.txt	6.25 KB (6,400 bytes)	223 bytes (223 bytes)	219 bytes (219 bytes)
output39s.txt	6.25 KB (6,400 bytes)	223 bytes (223 bytes)	214 bytes (214 bytes)
output40s.txt	6.25 KB (6,400 bytes)	223 bytes (223 bytes)	214 bytes (214 bytes)
output41s.txt	78.1 KB (80,000 bytes)	267 bytes (267 bytes)	372 bytes (372 bytes)
output42s.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	218 bytes (218 bytes)
output43s.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	219 bytes (219 bytes)
output44s.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	216 bytes (216 bytes)
output45s.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	219 bytes (219 bytes)
output46s.txt	76.2 MB (80,000,000 bytes)	11.2 KB (11,543 bytes)	168 KB (172,900 bytes)
output47s.txt	78.1 KB (80,000 bytes)	267 bytes (267 bytes)	371 bytes (371 bytes)
output48s.txt	78.1 KB (80,000 bytes)	267 bytes (267 bytes)	371 bytes (371 bytes)
output49s.txt	78.1 KB (80,000 bytes)	267 bytes (267 bytes)	373 bytes (373 bytes)
output50s.txt	78.1 KB (80,000 bytes)	267 bytes (267 bytes)	370 bytes (370 bytes)

Tabla 38: Archivos encriptados con el algoritmo original AES comprimidos con la clave aaaaaaaaaaaaaaaa (mejor caso)

Elaboración: Propia

Clave: 1111111111111111			
Nombre del archivo	**Tamaño del archivo**	**Tamaño de archivo comprimido (LZMA)**	**Tamaño de archivo comprimido (deflate)**
output1z.txt	38.1 MB (40,000,000 bytes)	5.76 KB (5,899 bytes)	84.5 KB (86,550 bytes)
output2z.txt	78.1 KB (80,000 bytes)	265 bytes (265 bytes)	370 bytes (370 bytes)
output3z.txt	78.1 KB (80,000 bytes)	265 bytes (265 bytes)	370 bytes (370 bytes)
output4z.txt	7.81 KB (8,000 bytes)	265 bytes (265 bytes)	369 bytes (369 bytes)
output5z.txt	78.1 KB (80,000 bytes)	265 bytes (265 bytes)	369 bytes (369 bytes)
output6z.txt	78.1 KB (80,000 bytes)	265 bytes (265 bytes)	369 bytes (369 bytes)
output7z.txt	78.1 KB (80,000 bytes)	265 bytes (265 bytes)	370 bytes (370 bytes)
output8z.txt	78.1 KB (80,000 bytes)	265 bytes (265 bytes)	371 bytes (371 bytes)
output9z.txt	78.1 KB (80,000 bytes)	265 bytes (265 bytes)	370 bytes (370 bytes)
output10z.txt	78.1 KB (80,000 bytes)	267 bytes (267 bytes)	370 bytes (370 bytes)
output11z.txt	78.1 KB (80,000 bytes)	267 bytes (267 bytes)	371 bytes (371 bytes)
output12z.txt	781 KB (800,000 bytes)	371 bytes (371 bytes)	1.88 KB (1,926 bytes)
output13z.txt	781 KB (800,000 bytes)	371 bytes (371 bytes)	1.88 KB (1,929 bytes)
output14z.txt	781 KB (800,000 bytes)	371 bytes (371 bytes)	1.88 KB (1,928 bytes)
output15z.txt	781 KB (800,000 bytes)	371 bytes (371 bytes)	1.88 KB (1,929 bytes)
output16z.txt	781 KB (800,000 bytes)	371 bytes (371 bytes)	1.88 KB (1,931 bytes)

output17z.txt	781 KB (800,000 bytes)	371 bytes (371 bytes)	1.88 KB (1,931 bytes)
output18z.txt	781 KB (800,000 bytes)	371 bytes (371 bytes)	1.88 KB (1,929 bytes)
output19z.txt	781 KB (800,000 bytes)	371 bytes (371 bytes)	1.88 KB (1,931 bytes)
output20z.txt	781 KB (800,000 bytes)	371 bytes (371 bytes)	1.88 KB (1,927 bytes)
output21z.txt	781 KB (800,000 bytes)	371 bytes (371 bytes)	1.88 KB (1,931 bytes)
output22z.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	220 bytes (220 bytes)
output23z.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	218 bytes (218 bytes)
output24z.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	217 bytes (217 bytes)
output25z.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	221 bytes (221 bytes)
output26z.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	217 bytes (217 bytes)
output27z.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	218 bytes (218 bytes)
output28z.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	220 bytes (220 bytes)
output29z.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	219 bytes (219 bytes)
output30z.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	219 bytes (219 bytes)
output31z.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	219 bytes (219 bytes)
output32z.txt	6.25 KB (6,400 bytes)	223 bytes (223 bytes)	216 bytes (216 bytes)
output33z.txt	6.25 KB (6,400 bytes)	223 bytes (223 bytes)	218 bytes (218 bytes)
output34z.txt	6.25 KB (6,400 bytes)	223 bytes (223 bytes)	216 bytes (216 bytes)

output35z.txt	6.25 KB (6,400 bytes)	223 bytes (223 bytes)	215 bytes (215 bytes)
output36z.txt	6.25 KB (6,400 bytes)	223 bytes (223 bytes)	218 bytes (218 bytes)
output37z.txt	6.25 KB (6,400 bytes)	223 bytes (223 bytes)	213 bytes (213 bytes)
output38z.txt	6.25 KB (6,400 bytes)	223 bytes (223 bytes)	218 bytes (218 bytes)
output39z.txt	6.25 KB (6,400 bytes)	223 bytes (223 bytes)	217 bytes (217 bytes)
output40z.txt	6.25 KB (6,400 bytes)	223 bytes (223 bytes)	218 bytes (218 bytes)
output41z.txt	78.1 KB (80,000 bytes)	267 bytes (267 bytes)	372 bytes (372 bytes)
output42z.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	214 bytes (214 bytes)
output43z.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	220 bytes (220 bytes)
output44z.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	219 bytes (219 bytes)
output45z.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	219 bytes (219 bytes)
output46z.txt	76.2 MB (80,000,000 bytes)	11.2 KB (11,543 bytes)	168 KB (172,901 bytes)
output47z.txt	78.1 KB (80,000 bytes)	267 bytes (267 bytes)	373 bytes (373 bytes)
output48z.txt	78.1 KB (80,000 bytes)	267 bytes (267 bytes)	373 bytes (373 bytes)
output49z.txt	78.1 KB (80,000 bytes)	267 bytes (267 bytes)	374 bytes (374 bytes)
output50z.txt	78.1 KB (80,000 bytes)	267 bytes (267 bytes)	372 bytes (372 bytes)

Tabla 39: Archivos encriptados con el algoritmo original AES comprimidos con la clave 1111111111111111 (mejor caso)

Elaboración: Propia

En este caso existe una gran disminución del tamaño de los archivos ya que Rijndael es un algoritmo de encriptación por bloques de 128 bits, debido a que el archivo a encriptar tiene caracteres repetitivos, el primer bloque se repetirá hasta cubrir la totalidad del archivo, al comprimir el archivo encriptado el resultado será igual a comprimir los primeros 128 bits multiplicado por el número de repeticiones de este hasta cubrir el tamaño total del archivo, sumando el tamaño los headers o cabeceras del algoritmo de compresión.

4.1.3. ANÁLISIS DE LA FUNCIÓN SHIFTROW DEL PROCESO, ENTRADAS Y SALIDAS

En el algoritmo Rijndael la función shiftrow opera las filas del estado, cambia cíclicamente los bytes en cada fila por un cierto desplazamiento. Para AES, la primera fila se deja sin cambios. Cada byte de la segunda fila se desplaza uno a la izquierda. De manera similar, la tercera y cuarta filas se desplazan por desplazamientos de dos y tres respectivamente, de la manera que sigue.

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$

ShiftRows ()
→

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$
$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,0}$
$a_{2,2}$	$a_{2,3}$	$a_{2,0}$	$a_{2,1}$
$a_{3,3}$	$a_{3,0}$	$a_{3,1}$	$a_{3,2}$

Tabla 40: Transformación ShiftRows

Elaboración: Propia

Para las claves de 128 bits se usan 10 rondas, 12 rondas para las claves de tamaño 192 y 14 para las claves de 256 bits. Por lo que los desplazamientos

en la función ShiftRows ignorando las demás trasformaciones quedarían de la siguiente manera.

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$

→

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$
$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,0}$
$a_{2,2}$	$a_{2,3}$	$a_{2,0}$	$a_{2,1}$
$a_{3,3}$	$a_{3,0}$	$a_{3,1}$	$a_{3,2}$

Tabla 41: Primera Ronda ShiftRows

Elaboración: Propia

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$
$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,0}$
$a_{2,2}$	$a_{2,3}$	$a_{2,0}$	$a_{2,1}$
$a_{3,3}$	$a_{3,0}$	$a_{3,1}$	$a_{3,2}$

→

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$
$a_{1,2}$	$a_{1,3}$	$a_{1,0}$	$a_{1,1}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$
$a_{3,2}$	$a_{3,3}$	$a_{3,0}$	$a_{3,1}$

Tabla 42: Segunda Ronda ShiftRows

Elaboración: Propia

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$
$a_{1,2}$	$a_{1,3}$	$a_{1,0}$	$a_{1,1}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$
$a_{3,2}$	$a_{3,3}$	$a_{3,0}$	$a_{3,1}$

→

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$
$a_{1,3}$	$a_{1,0}$	$a_{1,1}$	$a_{1,2}$
$a_{2,2}$	$a_{2,3}$	$a_{2,0}$	$a_{2,1}$
$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,0}$

Tabla 43: Tercera Ronda ShiftRows

Elaboración: Propia

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$
$a_{1,3}$	$a_{1,0}$	$a_{1,1}$	$a_{1,2}$
$a_{2,2}$	$a_{2,3}$	$a_{2,0}$	$a_{2,1}$
$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,0}$

→

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$

Tabla 44: Cuarta Ronda ShiftRows

Elaboración: Propia

De estas tablas podemos apreciar que:

- La ronda 4, la función ShiftRows regresa a la matriz a su posición inicial
- La ronda 10, las posiciones de la matriz obtenida son las misma que en la segunda ronda
- La ronda 12, las posiciones de la matriz obtenida es igual que la matriz inicial.
- La ronda 14, las posiciones de la matriz obtenida son las mismas que en la segunda ronda.

Tenemos 2 posibles estados finales para cualquier número de rondas (Nr) utilizados en el algoritmo Rijndael.

4.2. DISEÑO DE LAS MODIFICACIONES DEL ALGORITMO RIJNDAEL

Para realizar el estudio del impacto de la modificación del AES se modificó el algoritmo original de la siguiente forma:

4.2.1. MODIFICACIÓN Y ANÁLISIS DE LA FUNCIÓN KEYEXPANSION

Además de expandir las claves calculamos los movimientos que realizaremos en ShiftRow sumando los elementos de las filas de cada clave expandida y aplicándoles módulo 2 de manera que obtendremos un array de 1 ó 0 que llamaremos ShiftMoves.

Clave de cifrado				Sub clave de ronda 1				Sub clave de ronda 2				Sub clave de ronda 3					Sub clave de ronda Nr			
$2b$	28	ab	09	$a0$	88	23	$2a$	$f2$	$7a$	59	73	$3d$	47	$1e$	$6d$		$d0$	$c9$	$e1$	$b6$
$7e$	ae	$f7$	cf	fa	54	$a3$	$6c$	$c2$	96	35	59	80	16	23	$7a$		14	ee	$3f$	63
15	$d2$	15	$4f$	fe	$2c$	39	76	95	$b9$	80	$f6$	47	fe	$7e$	88	…	$f9$	25	$0c$	$0c$
16	$a6$	88	$3c$	17	$b1$	39	05	$f2$	43	$7a$	$7f$	$7d$	$3e$	44	$3b$		$a8$	89	$c8$	$a6$

Tabla 45: Subclaves de la función KeyExpansion

Fuente Elaboración Propia

Sumando los elementos de las filas de la primera sub-clave:

$$(a0 + 88 + 23 + 2a)Mod\ 2 = 1$$

$$(fa + 54 + a3 + 6c)Mod2 = 1$$

$$(fe + 2c + 39 + 76)Mod2 = 1$$

$$(17 + b1 + 39 + 05)Mod2 = 0$$

Guardamos estos resultados en el array ShiftMoves por lo que en este tendremos elementos, que definirán la dirección de las movidas en la función ShiftRows.

4.2.2. MODIFICACIÓN Y ANÁLISIS DE LA FUNCIÓN SHIFTROW

En esta sección describiremos las principales modificaciones propuestas para el algoritmo AES y analizaremos cuales son las posibles principales salidas del algoritmo modificado.

- Conservaremos la primera fila intacta como en el algoritmo original,
- Para la segunda fila, verificamos ShiftMoves[(4*round)-3], en caso de ser 0 el movimiento de la fila en ShiftRow () será a la izquierda en caso de ser 1 a la derecha.
- Para la tercera fila, verificamos ShiftMoves[(4*round)-2], en caso de ser 0 el movimiento de la fila en ShiftRow () será a la izquierda en caso de ser 1 a la derecha.
- Para la última fila, verificamos ShiftMoves[(4*round)-1], en caso de ser 0 el movimiento de la fila en ShiftRow () será a la izquierda en caso de ser 1 a la derecha.

La modificación de esta función nos da cuatro posibles salidas para cada ronda.

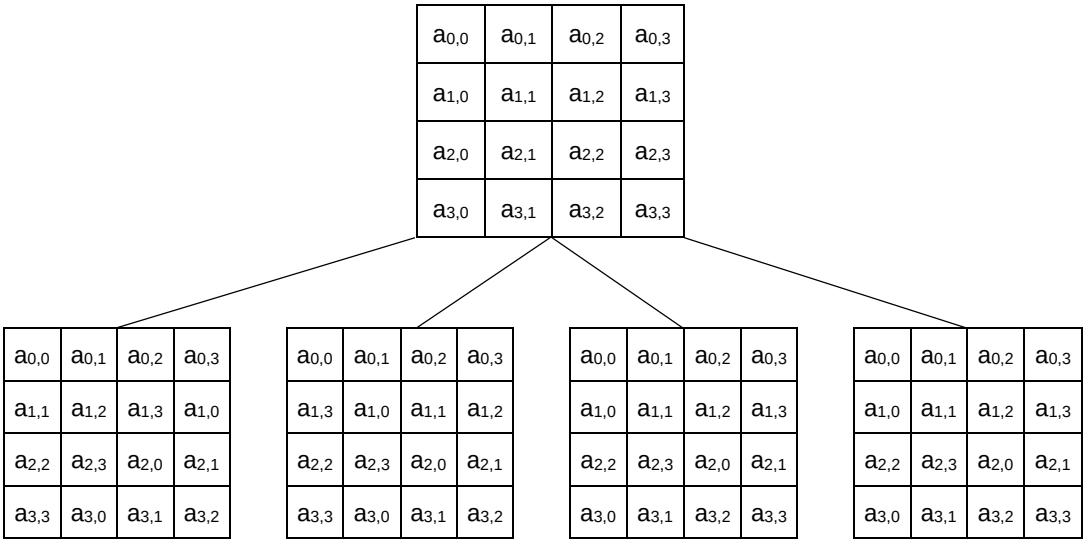


Tabla 46: Posibles salidas para la primera ronda de ShiftRows

Elaboración: Propia

4.2.3. ANÁLISIS DE LA MODIFICACIÓN EN LA FUNCIÓN SHIFTROWS DEL PROCESO, ENTRADAS Y SALIDAS

La modificación en la función ShiftRows del algoritmo Rijndael opera las filas de la matriz de estado; cambia cíclicamente los bytes en cada fila por un desplazamiento definido en la función KeyExpansion. Para esta modificación, igual que en el algoritmo original.

- La primera fila se deja sin cambios
- En la segunda fila, cada byte se desplaza una posición a la izquierda o derecha, la dirección de este desplazamiento está definido por la variable ShiftMoves.
- En la tercera fila, cada byte se desplaza 2 posiciones a la izquierda o derecha, la dirección de este desplazamiento también está definida en la variable ShiftMoves.

- En la cuarta fila, cada byte se desplaza 3 posiciones a la izquierda o derecha, la dirección de este desplazamiento también está definida en la variable ShiftMoves.

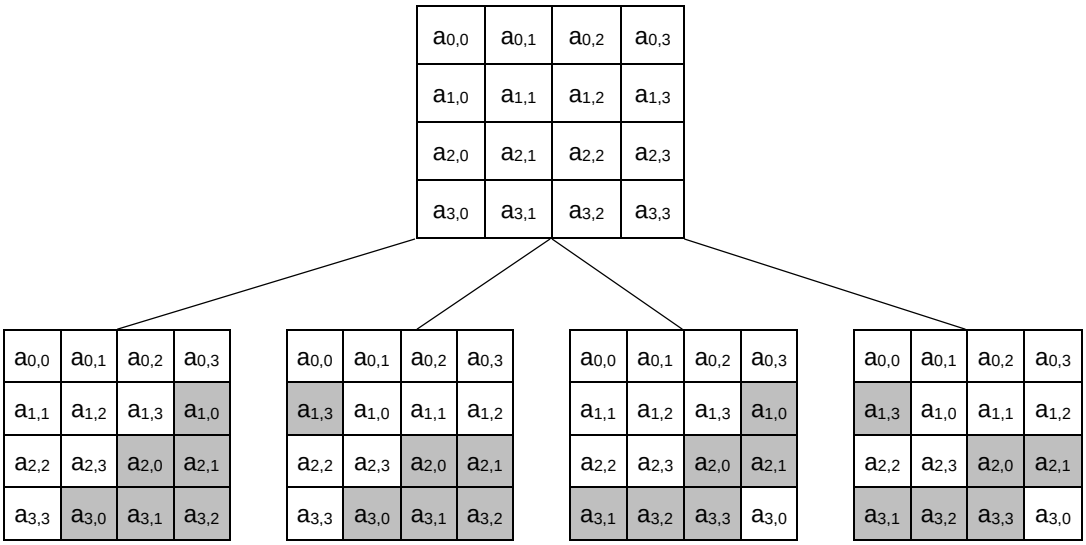


Tabla 47: Posibles matrices finales para la modificación de la función ShiftRows

Elaboración: Propia

Tememos cuatros posibles estados finales después de la primera ronda, después de la segunda tendremos 16 posibles estados, podemos deducir que los posibles estados finales están dado en razón de 4^{Nr}, donde Nr es el número de rondas del algoritmo.

4.3. IMPLEMENTACIÓN Y ANÁLISIS DE LA MODIFICACIÓN DEL ALGORITMO RIJNDAEL.

En este capítulo se describe la implementación del nuevo algoritmo para las funciones KeyExpansion y ShiftRows planteado en el sub capítulo 42, se implementó el algoritmo utilizando el lenguaje c++, también se realizó el análisis

de la complejidad temporal utilizando la notación big O ó O mayúscula y posterior a esto se comprimieron los archivos encriptados, de esta manera podremos medir de la entropía de las salidas.

4.3.1. ANÁLISIS ASINTÓTICO DE LA MODIFICACIÓN DE LA FUNCIÓN KEYEXPANSION

Basándonos en la modificación planteada en el sub capítulo 4.2.1 se plantea que el nuevo algoritmo KeyExpansion sea capaz de sumar los cuatro elementos de cada fila de la matriz de estado de cada subclave generada, a este resultado se debe aplicar la operación Mod 2 (%2), para almacenarlo en un array ShiftMoves;

$k_{0,0}$	$k_{0,1}$	$k_{0,2}$	$k_{0,3}$
$k_{1,0}$	$k_{1,1}$	$k_{1,2}$	$k_{1,3}$
$k_{2,0}$	$k_{2,1}$	$k_{2,2}$	$k_{2,3}$
$k_{3,0}$	$k_{3,1}$	$k_{3,2}$	$k_{3,3}$

Tabla 48: Matriz de subclaves

Elaboración: Propia

$$Sm_0 = (k_{0,0}, k_{0,1}, k_{0,2}, k_{0,3}) Mod\ 2 \qquad Sm_2 = (k_{2,0}, k_{2,1}, k_{2,2}, k_{2,3}) Mod\ 2$$

$$Sm_1 = (k_{1,0}, k_{1,1}, k_{1,2}, k_{1,3}) Mod\ 2 \qquad Sm_3 = (k_{3,0}, k_{3,1}, k_{3,2}, k_{3,3}) Mod\ 2$$

$$ShiftMoves[Nr * 4] = [Sm_0, Sm_1, Sm_2, Sm_3, \dots, S_{Nr*4-1}]$$

Por lo que la implementación final en c++ de la modificación de la función KeyExpansion estaría dada por:

```
KeyExpansion () {                                                    
  int i,j;                                                           
  unsigned char temp[4],k;                                           

  for (i=0; i<Nk; i++){                                              O(Nk)(
    RoundKey[i*4]=Key[i*4];                                           O(1)
    RoundKey[i*4+1]=Key[i*4+1];                                       O(1)
    RoundKey[i*4+2]=Key[i*4+2];                                       O(1)
    RoundKey[i*4+3]=Key[i*4+3];                                       O(1)
  }                                                                  )

  while (i < (Nb * (Nr+1))){                                         O(Nb * (Nr + 1))(
    for(j=0;j<4;j++){                                                 O(4)(
      temp[j]=RoundKey[(i-1) * 4 + j];                                 O(1)
    }                                                                 )

    if (i % Nk == 0){                                                 O(1) +
      {// Función RotWord ();
        k = temp[0];                                                   O(1/Nk)
        temp[0] = temp[1];                                             O(1/Nk)
        temp[1] = temp[2];                                             O(1/Nk)
        temp[2] = temp[3];                                             O(1/Nk)
        temp[3] = k;                                                   O(1/Nk)
      }
      {// Función SubWord ();
        temp[0]=getSBoxValue(temp[0]);                                 O(getSBoxValue/Nk)
        temp[1]=getSBoxValue(temp[1]);                                 O(getSBoxValue/Nk)
        temp[2]=getSBoxValue(temp[2]);                                 O(getSBoxValue/Nk)
        temp[3]=getSBoxValue(temp[3]);                                 O(getSBoxValue/Nk)
      }
      temp[0] =  temp[0] ^ Rcon[i/Nk];                                 O(1/Nk)
    }
    else if (Nk > 6 && i % Nk == 4){                                   O(1) +
      {// Función Subword()
        temp[0]=getSBoxValue(temp[0]);                                 O(getSBoxValue/Nk)
        temp[1]=getSBoxValue(temp[1]);                                 O(getSBoxValue/Nk)
        temp[2]=getSBoxValue(temp[2]);                                 O(getSBoxValue/Nk)
        temp[3]=getSBoxValue(temp[3]);                                 O(getSBoxValue/Nk)
      }
    }
    RoundKey[i*4+0] = RoundKey[(i-Nk)*4+0] ^ temp[0];                  O(1)
    RoundKey[i*4+1] = RoundKey[(i-Nk)*4+1] ^ temp[1];                  O(1)
    RoundKey[i*4+2] = RoundKey[(i-Nk)*4+2] ^ temp[2];                  O(1)
    RoundKey[i*4+3] = RoundKey[(i-Nk)*4+3] ^ temp[3];                  O(1)
    ShiftMoves[i-Nk]   =   (RoundKey[i*4+0]   +   RoundKey[i*4+1]   +
RoundKey[i*4+2] + RoundKey[i*4+3]) % 2;                                O(1)
  i++;                                                                 O(1)
  }                                                                   )
}
```

El análisis asintótico de la función KeyExpansion () nos da la ecuación:

$$O(Nk) + O\Bigg(Nb * (Nr + 1)\left(O(4) + O\left(\frac{5 + 4(\text{getSBoxValue})}{Nk}\right) + O\left(\frac{\text{getSBoxValue}}{4}\right) + O(5)\right)\Bigg)$$

Donde:

$$Nr: es\ el\ número\ de\ rondas\ (10{,}12\ y\ 14).$$

$$Nk: Número\ de\ palabras\ que\ comprenden\ la\ clave\ de\ cifrado.\ (4, 6\ o\ 8).$$

$$Nb: Número\ de\ columnas\ es\ 4\ para\ todos\ los\ casos.$$

Estos valores se desestiman ya que no representan mayor complejidad ejecutarlos; la función getSBoxValue() es una asignación simple por lo que la ecuación también se reduce a:

$$O(k) + O\Bigg(4 * (k + 1)\left(O(4) + O\left(\frac{5 + 4(\text{getSBoxValue})}{k}\right) + O\left(\frac{\text{getSBoxValue}}{4}\right) + O(5)\right)\Bigg)$$

Reduciendo la ecuación quedaría como

$$O\left(\frac{(\text{getSBoxValue})}{k}\right) + O\left(\frac{\text{getSBoxValue}}{4}\right)$$

La función getSBoxValue no se modifica por lo que su tiempo asintótico al ejecutarse sigue siendo el mismo $O(k)$ por lo que podemos asumir que la ecuación asintótica final para la modificación de función KeyExpansion es:

$$O(k')$$

Con lo que demostramos que la modificación de la función KeyExpansion no tiene mayor complejidad temporal que el original

4.3.2. ANÁLISIS ASINTÓTICO DE LA MODIFICACIÓN DE LA FUNCIÓN SHIFTROWS

La nueva función ShiftRows debería poder leer la variable ShiftMoves para identificar la dirección en la que tiene que mover cada fila de la matriz de estado y realizar los movimientos a la matriz de estado, el código final quedara como:

```
ShiftRows(int round){                                   
 char temp;                                             
                                                        
 if(ShiftMoves[(4*round)-3]==0){                        O(1) + (
  temp=state[1][0];                                      O(1)
  state[1][0]=state[1][1];                               O(1)
  state[1][1]=state[1][2];                               O(1)
  state[1][2]=state[1][3];                               O(1)
  state[1][3]=temp;                                      O(1)
 }                                                      )
 else if(ShiftMoves[(4*round)-3]==1){                   O(1) + (
  temp=state[1][3];                                      O(1)
  state[1][3]=state[1][2];                               O(1)
  state[1][2]=state[1][1];                               O(1)
  state[1][1]=state[1][0];                               O(1)
  state[1][0]=temp;                                      O(1)
 }                                                      )
 // Rotate second row 2 columns to left or right
 temp=state[2][0];                                      O(1)
 state[2][0]=state[2][2];                               O(1)
 state[2][2]=temp;                                      O(1)
 temp=state[2][1];                                      O(1)
 state[2][1]=state[2][3];                               O(1)
 state[2][3]=temp;                                      O(1)

 if(ShiftMoves[(4*round)-2]==0){                        O(1) + (
  temp=state[3][0];                                      O(1)
```

```
  state[3][0]=state[3][3];                                  O(1)
  state[3][3]=state[3][2];                                  O(1)
  state[3][2]=state[3][1];                                  O(1)
  state[3][1]=temp;                                         O(1)
 }                                                          )
 else if(ShiftMoves[(4*round)-2]==1){                       O(1) + (
  temp=state[3][0];                                         O(1)
  state[3][0]=state[3][1];                                  O(1)
  state[3][1]=state[3][2];                                  O(1)
  state[3][2]=state[3][3];                                  O(1)
  state[3][3]=temp;                                         O(1)
 }                                                          )
}
```

La ecuación para definir la complejidad temporal de la nueva función ShiftRows estaría definida por:

$$O(1) + O(5) + O(1) + O(5) + O(6) + O(1) + O(5) + O(1) + O(5)$$

Por ser números naturales la ecuación se puede resumir a:

$$\mathrm{O}(k')$$

Donde k es una constante.

Ya no se realizó el análisis asintótico de las demás funciones por que no se modificaron, el análisis asintótico es el mismo, por lo que la ecuación general del algoritmo quedaría dada por:

$$\mathrm{O}(\mathrm{KeyExpansion}) + \mathrm{O}(AddRoundkey) + \mathrm{O}\Big(Nr\big(\mathrm{O}(SubBytes) + \mathrm{O}(ShiftRows) + \mathrm{O}(MixColumns) + \mathrm{O}(AddRoundKey)\big)\Big) + \mathrm{O}(\mathrm{subBytes}) + \mathrm{O}(\mathrm{ShiftRows}) + \mathrm{O}(\mathrm{AddRoundkey})$$

Reduciendo la ecuacion:

$$\mathrm{O}(KeyExpansion) + \mathrm{O}(SubBytes) + \mathrm{O}(ShiftRow) + \mathrm{O}(MixColumn) + \mathrm{O}(AddRoundKey)$$

Reemplazando

$$O(k) + O(k) + O(k') + O(k) + O(k')$$

Esta ecuación se puede reducir a:

$$O(k)$$

4.3.3. ENTROPÍA DE LAS SALIDAS DEL ALGORITMO AES MODIFICADO.

Se implementó la modificación del algoritmo en el lenguaje c++, se encripto los mismos archivos de texto utilizados con el algoritmo original, con las mismas claves utilizadas anteriormente, luego se procedió a comprimir dichos archivos con el fin de encontrar dependencia matemática entre sus caracteres, para así tener una medida del impacto de la modificación de las funciones en las salidas del algoritmo.

4.3.3.1. PEOR CASO

Para las pruebas con el peor caso para ambos algoritmos se consideró una clave fuerte como 1Ds$f23-USAbCc12, y archivos con caracteres aleatorios de diferentes tamaño, para de esta forma crear las condiciones óptimas para observar los efectos del algoritmo en archivos aleatorios.

Se utilizó la siguiente configuración con el programa 7-zip y diccionarios para los diferentes tamaños de archivo, y diferentes tipos de compresión tanto para los archivos encriptados, tanto para los archivos generados con la clave 1 como para la clave 2.

Formato	Zip	**Tipo de compresión**	LZMA

Nivel de compresión	Ultra	**Tamaño de palabra**	8
Numero de hilos de la CPU	8		
Archivos	**Tamaño**	**Diccionario**	
output1.txt	762 MB	256 MB	
output2.txt – output11.txt	625 KB	1 MB	
output12.txt – output15.txt	1.98 MB	2 MB	
output16.txt – output25.txt	3.05 MB	3 MB	
output26.txt – output30.txt	305 MB	256 MB	
output31.txt – output40.txt	457 MB	256 MB	
output41.txt – output45.txt	7.81 KB	64 KB	
output46.txt – output50.txt	76.2 MB	96 MB	

Tabla 49: Configuración LZMA utilizada para comprimir los archivos generados para el peor caso con el algoritmo modificado.

Elaboracion: Propia

Formato	Zip	**Tipo de compresión**	Deflate
Nivel de compresión	Ultra	**Tamaño de palabra**	8
Numero de hilos de la CPU	8	**Tamaño de diccionario**	32 KB

Tabla 50: Configuración Deflate utilizada para comprimir los archivos generados para el peor caso con el algoritmo modificado.

Elaboración: Propia

Clave: 1Ds$f23-USAbCc12			
Nombre del archivo	**Tamaño del archivo**	**Tamaño de archivo comprimido (LZMA)**	**Tamaño de archivo comprimido (Deflate)**
output1.txt	762 MB (800,000,000 bytes)	762 MB (800,000,156 bytes)	762 MB (800,000,156 bytes)
output2.txt	625 KB (640,000 bytes)	625 KB (640,156 bytes)	625 KB (640,156 bytes)
output3.txt	625 KB (640,000 bytes)	625 KB (640,156 bytes)	625 KB (640,156 bytes)
output4.txt	625 KB (640,000 bytes)	625 KB (640,156 bytes)	625 KB (640,156 bytes)
output5.txt	625 KB (640,000 bytes)	625 KB (640,156 bytes)	625 KB (640,156 bytes)
output6.txt	625 KB (640,000 bytes)	625 KB (640,156 bytes)	625 KB (640,156 bytes)
output7.txt	625 KB (640,000 bytes)	625 KB (640,156 bytes)	625 KB (640,156 bytes)
output8.txt	625 KB (640,000 bytes)	625 KB (640,156 bytes)	625 KB (640,156 bytes)
output9.txt	625 KB (640,000 bytes)	625 KB (640,156 bytes)	625 KB (640,156 bytes)
output10.txt	625 KB (640,000 bytes)	625 KB (640,158 bytes)	625 KB (640,158 bytes)
output11.txt	625 KB (640,000 bytes)	625 KB (640,158 bytes)	625 KB (640,158 bytes)
output12.txt	1.98 MB (2,080,000 bytes)	1.98 MB (2,080,158 bytes)	1.98 MB (2,080,158 bytes)
output13.txt	1.98 MB (2,080,000 bytes)	1.98 MB (2,080,158 bytes)	1.98 MB (2,080,158 bytes)
output14.txt	1.98 MB (2,080,000 bytes)	1.98 MB (2,080,158 bytes)	1.98 MB (2,080,158 bytes)
output15.txt	1.98 MB (2,080,000 bytes)	1.98 MB (2,080,158 bytes)	1.98 MB (2,080,158 bytes)
output16.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,158 bytes)	3.05 MB (3,200,158 bytes)

output17.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,158 bytes)	3.05 MB (3,200,158 bytes)
output18.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,158 bytes)	3.05 MB (3,200,158 bytes)
output19.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,158 bytes)	3.05 MB (3,200,158 bytes)
output20.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,158 bytes)	3.05 MB (3,200,158 bytes)
output21.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,158 bytes)	3.05 MB (3,200,158 bytes)
output22.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,158 bytes)	3.05 MB (3,200,158 bytes)
output23.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,158 bytes)	3.05 MB (3,200,158 bytes)
output24.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,158 bytes)	3.05 MB (3,200,158 bytes)
output25.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,158 bytes)	3.05 MB (3,200,158 bytes)
output26.txt	305 MB (320,000,000 bytes)	305 MB (320,000,158 bytes)	305 MB (320,000,158 bytes)
output27.txt	305 MB (320,000,000 bytes)	305 MB (320,000,158 bytes)	305 MB (320,000,158 bytes)
output28.txt	305 MB (320,000,000 bytes)	305 MB (320,000,158 bytes)	305 MB (320,000,158 bytes)
output29.txt	305 MB (320,000,000 bytes)	305 MB (320,000,158 bytes)	305 MB (320,000,158 bytes)
output30.txt	305 MB (320,000,000 bytes)	305 MB (320,000,158 bytes)	305 MB (320,000,158 bytes)
output31.txt	457 MB (480,000,000 bytes)	457 MB (480,000,158 bytes)	457 MB (480,000,158 bytes)
output32.txt	457 MB (480,000,000 bytes)	457 MB (480,000,158 bytes)	457 MB (480,000,158 bytes)
output33.txt	457 MB (480,000,000 bytes)	457 MB (480,000,158 bytes)	457 MB (480,000,158 bytes)
output34.txt	457 MB (480,000,000 bytes)	457 MB (480,000,158 bytes)	457 MB (480,000,158 bytes)

output35.txt	457 MB (480,000,000 bytes)	457 MB (480,000,158 bytes)	457 MB (480,000,158 bytes)
output36.txt	457 MB (480,000,000 bytes)	457 MB (480,000,158 bytes)	457 MB (480,000,158 bytes)
output37.txt	457 MB (480,000,000 bytes)	457 MB (480,000,158 bytes)	457 MB (480,000,158 bytes)
output38.txt	457 MB (480,000,000 bytes)	457 MB (480,000,158 bytes)	457 MB (480,000,158 bytes)
output39.txt	457 MB (480,000,000 bytes)	457 MB (480,000,158 bytes)	457 MB (480,000,158 bytes)
output40.txt	457 MB (480,000,000 bytes)	457 MB (480,000,158 bytes)	457 MB (480,000,158 bytes)
output41.txt	7.81 KB (8,000 bytes)	7.96 KB (8,158 bytes)	7.96 KB (8,158 bytes)
output42.txt	7.81 KB (8,000 bytes)	7.96 KB (8,158 bytes)	7.96 KB (8,158 bytes)
output43.txt	7.81 KB (8,000 bytes)	7.96 KB (8,158 bytes)	7.96 KB (8,158 bytes)
output44.txt	7.81 KB (8,000 bytes)	7.96 KB (8,158 bytes)	7.96 KB (8,158 bytes)
output45.txt	7.81 KB (8,000 bytes)	7.96 KB (8,158 bytes)	7.96 KB (8,158 bytes)
output46.txt	76.2 MB (80,000,000 bytes)	76.2 MB (80,000,158 bytes)	76.2 MB (80,000,158 bytes)
output47.txt	76.2 MB (80,000,000 bytes)	76.2 MB (80,000,158 bytes)	76.2 MB (80,000,158 bytes)
output48.txt	76.2 MB (80,000,000 bytes)	76.2 MB (80,000,158 bytes)	76.2 MB (80,000,158 bytes)
output49.txt	76.2 MB (80,000,000 bytes)	76.2 MB (80,000,158 bytes)	76.2 MB (80,000,158 bytes)
output50.txt	76.2 MB (80,000,000 bytes)	76.2 MB (80,000,158 bytes)	76.2 MB (80,000,158 bytes)

Tabla 51: Archivos encriptados con el algoritmo modificado AES comprimidos con la clave 1Ds$f23-USAbCc12 (peor caso)

Elaboracion: Propia

Clave: abcdefghijklmnop			
Nombre del archivo	**Tamaño del archivo**	**Tamaño de archivo comprimido (LZMA)**	**Tamaño de archivo comprimido (Deflate)**
output1b.txt	762 MB (800,000,000 bytes)	762 MB (800,000,158 bytes)	762 MB (800,000,158 bytes)
output2b.txt	625 KB (640,000 bytes)	625 KB (640,158 bytes)	625 KB (640,158 bytes)
output3b.txt	625 KB (640,000 bytes)	625 KB (640,158 bytes)	625 KB (640,158 bytes)
output4b.txt	625 KB (640,000 bytes)	625 KB (640,158 bytes)	625 KB (640,158 bytes)
output5b.txt	625 KB (640,000 bytes)	625 KB (640,158 bytes)	625 KB (640,158 bytes)
output6b.txt	625 KB (640,000 bytes)	625 KB (640,158 bytes)	625 KB (640,158 bytes)
output7b.txt	625 KB (640,000 bytes)	625 KB (640,158 bytes)	625 KB (640,158 bytes)
output8b.txt	625 KB (640,000 bytes)	625 KB (640,158 bytes)	625 KB (640,158 bytes)
output9b.txt	625 KB (640,000 bytes)	625 KB (640,158 bytes)	625 KB (640,158 bytes)
output10b.txt	625 KB (640,000 bytes)	625 KB (640,158 bytes)	625 KB (640,160 bytes)
output11b.txt	625 KB (640,000 bytes)	625 KB (640,158 bytes)	625 KB (640,160 bytes)
output12b.txt	1.98 MB (2,080,000 bytes)	1.98 MB (2,080,160 bytes)	1.98 MB (2,080,160 bytes)
output13b.txt	1.98 MB (2,080,000 bytes)	1.98 MB (2,080,160 bytes)	1.98 MB (2,080,160 bytes)
output14b.txt	1.98 MB (2,080,000 bytes)	1.98 MB (2,080,160 bytes)	1.98 MB (2,080,160 bytes)
output15b.txt	1.98 MB (2,080,000 bytes)	1.98 MB (2,080,160 bytes)	1.98 MB (2,080,160 bytes)
output16b.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,160 bytes)	3.05 MB (3,200,160 bytes)
output17b.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,160 bytes)	3.05 MB (3,200,160 bytes)
output18b.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,160 bytes)	3.05 MB (3,200,160 bytes)
output19b.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,160 bytes)	3.05 MB (3,200,160 bytes)
output20b.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,160 bytes)	3.05 MB (3,200,160 bytes)

output21b.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,160 bytes)	3.05 MB (3,200,160 bytes)
output22b.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,160 bytes)	3.05 MB (3,200,160 bytes)
output23b.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,160 bytes)	3.05 MB (3,200,160 bytes)
output24b.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,160 bytes)	3.05 MB (3,200,160 bytes)
output25b.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,160 bytes)	3.05 MB (3,200,160 bytes)
output26b.txt	305 MB (320,000,000 bytes)	305 MB (320,000,160 bytes)	305 MB (320,000,160 bytes)
output27b.txt	305 MB (320,000,000 bytes)	305 MB (320,000,160 bytes)	305 MB (320,000,160 bytes)
output28b.txt	305 MB (320,000,000 bytes)	305 MB (320,000,160 bytes)	305 MB (320,000,160 bytes)
output29b.txt	305 MB (320,000,000 bytes)	305 MB (320,000,160 bytes)	305 MB (320,000,160 bytes)
output30b.txt	305 MB (320,000,000 bytes)	305 MB (320,000,160 bytes)	305 MB (320,000,160 bytes)
output31b.txt	457 MB (480,000,000 bytes)	457 MB (480,000,160 bytes)	457 MB (480,000,160 bytes)
output32b.txt	457 MB (480,000,000 bytes)	457 MB (480,000,160 bytes)	457 MB (480,000,160 bytes)
output33b.txt	457 MB (480,000,000 bytes)	457 MB (480,000,160 bytes)	457 MB (480,000,160 bytes)
output34b.txt	457 MB (480,000,000 bytes)	457 MB (480,000,160 bytes)	457 MB (480,000,160 bytes)
output35b.txt	457 MB (480,000,000 bytes)	457 MB (480,000,160 bytes)	457 MB (480,000,160 bytes)
output36b.txt	457 MB (480,000,000 bytes)	457 MB (480,000,160 bytes)	457 MB (480,000,160 bytes)
output37b.txt	457 MB (480,000,000 bytes)	457 MB (480,000,160 bytes)	457 MB (480,000,160 bytes)
output38b.txt	457 MB (480,000,000 bytes)	457 MB (480,000,160 bytes)	457 MB (480,000,160 bytes)
output39b.txt	457 MB (480,000,000 bytes)	457 MB (480,000,160 bytes)	457 MB (480,000,160 bytes)
output40b.txt	457 MB (480,000,000 bytes)	457 MB (480,000,160 bytes)	457 MB (480,000,160 bytes)
output41b.txt	7.81 KB (8,000 bytes)	7.96 KB (8,160 bytes)	7.96 KB (8,160 bytes)
output42b.txt	7.81 KB (8,000 bytes)	7.96 KB (8,160 bytes)	7.96 KB (8,160 bytes)
output43b.txt	7.81 KB (8,000 bytes)	7.96 KB (8,160 bytes)	7.96 KB (8,160 bytes)
output44b.txt	7.81 KB (8,000 bytes)	7.96 KB (8,160 bytes)	7.96 KB (8,160 bytes)

output45b.txt	7.81 KB (8,000 bytes)	7.96 KB (8,160 bytes)	7.96 KB (8,160 bytes)
output46b.txt	76.2 MB (80,000,000 bytes)	76.2 MB (80,000,160 bytes)	76.2 MB (80,000,160 bytes)
output47b.txt	76.2 MB (80,000,000 bytes)	76.2 MB (80,000,160 bytes)	76.2 MB (80,000,160 bytes)
output48b.txt	76.2 MB (80,000,000 bytes)	76.2 MB (80,000,160 bytes)	76.2 MB (80,000,160 bytes)
output49b.txt	76.2 MB (80,000,000 bytes)	76.2 MB (80,000,160 bytes)	76.2 MB (80,000,160 bytes)
output50b.txt	76.2 MB (80,000,000 bytes)	76.2 MB (80,000,160 bytes)	76.2 MB (80,000,160 bytes)

Tabla 52: Archivos encriptados con el algoritmo modificado AES comprimidos con la clave abcdefghijklmnop (peor caso)

Elaboración: Propia

Al comprimir los archivos estos aumentan su tamaño ya que las salidas del algoritmo Rijndael son muy independientes matemáticamente, y al someter el archivo encriptado a este procedimiento se agregan identificadores o headers propios del algoritmo de compresión, es por esto que incrementa su tamaño. Podemos notar también que la modificación no disminuye la independencia matemática.

4.3.3.2. MEJOR CASO

Para el mejor caso se tomó la clave **aaaaaaaaaaaaaaaa** y se utilizó los mismos archivos generados para el algoritmo original en el sub capítulo 4.1.2.2, estas entradas son las entradas más simples que se le puede dar al algoritmo original y a la modificación de este, con estas entradas se esperó que los archivos encriptados tengan una menor entropía en sus salidas ya que se repetirá el proceso de encriptación hasta cubrir el tamaño total del archivo.

Se utilizó la siguiente configuración con el programa 7-zip y diccionarios para los diferentes tamaños de archivo encriptados, y diferentes tipos de compresión tanto para los archivos encriptados.

Formato	Zip	**Tipo de compresión**	LZMA
Nivel de compresión	Ultra	**Tamaño de palabra**	8
Numero de hilos de la CPU	8		

Archivos	**Tamaño**	**Diccionario**
output1s.txt	38.1 MB	48 MB
output2s.txt – output11s.txt	78.1 KB	64 KB
output12.txt – output21.txt	781 KB	1 MB
output22.txt – output31.txt	7.81 KB	64 KB
output32.txt – output40.txt	6.25 KB	64 KB
output41.txt	78.1 KB	64 KB
output42.txt – output45.txt	7.81 KB	64 KB
output46.txt	76.2 MB	96 MB
output47.txt – output50.txt	78.1 KB	64 KB

Tabla 53: Configuración LZMA utilizada para comprimir los archivos generados para el mejor caso con el algoritmo modificado.

Elaboración: Propia

Formato	Zip	**Tipo de compresión**	Deflate
Nivel de compresión	Ultra	**Tamaño de palabra**	8
Numero de hilos de la CPU	8	**Tamaño de diccionario**	32 KB

Tabla 54: Configuración Deflate utilizada para comprimir los archivos generados para el mejor caso con el algoritmo modificado.

Elaboración: Propia

Clave: aaaaaaaaaaaaaaaa			
Nombre del archivo	**Tamaño del archivo encriptado**	**Tamaño de archivo comprimido (LZMA)**	**Tamaño de archivo comprimido (deflate)**
output1s.txt	38.1 MB (40,000,000 bytes)	5.76 KB (5,899 bytes)	84.5 KB (86,550 bytes)
output2s.txt	78.1 KB (80,000 bytes)	265 bytes (265 bytes)	370 bytes (370 bytes)
output3s.txt	78.1 KB (80,000 bytes)	265 bytes (265 bytes)	369 bytes (369 bytes)
output4s.txt	78.1 KB (80,000 bytes)	265 bytes (265 bytes)	372 bytes (372 bytes)
output5s.txt	78.1 KB (80,000 bytes)	265 bytes (265 bytes)	368 bytes (368 bytes)
output6s.txt	78.1 KB (80,000 bytes)	265 bytes (265 bytes)	372 bytes (372 bytes)
output7s.txt	78.1 KB (80,000 bytes)	265 bytes (265 bytes)	370 bytes (370 bytes)
output8s.txt	78.1 KB (80,000 bytes)	265 bytes (265 bytes)	371 bytes (371 bytes)
output9s.txt	78.1 KB (80,000 bytes)	265 bytes (265 bytes)	369 bytes (369 bytes)
output10s.txt	78.1 KB (80,000 bytes)	267 bytes (267 bytes)	369 bytes (369 bytes)

output11s.txt	78.1 KB (80,000 bytes)	267 bytes (267 bytes)	372 bytes (372 bytes)
output12s.txt	781 KB (800,000 bytes)	371 bytes (371 bytes)	1.88 KB (1,926 bytes)
output13s.txt	781 KB (800,000 bytes)	371 bytes (371 bytes)	1.88 KB (1,928 bytes)
output14s.txt	781 KB (800,000 bytes)	371 bytes (371 bytes)	1.88 KB (1,927 bytes)
output15s.txt	781 KB (800,000 bytes)	371 bytes (371 bytes)	1.88 KB (1,930 bytes)
output16s.txt	781 KB (800,000 bytes)	371 bytes (371 bytes)	1.88 KB (1,930 bytes)
output17s.txt	781 KB (800,000 bytes)	371 bytes (371 bytes)	1.88 KB (1,931 bytes)
output18s.txt	781 KB (800,000 bytes)	371 bytes (371 bytes)	1.88 KB (1,932 bytes)
output19s.txt	781 KB (800,000 bytes)	371 bytes (371 bytes)	1.88 KB (1,931 bytes)
output20s.txt	781 KB (800,000 bytes)	371 bytes (371 bytes)	1.88 KB (1,929 bytes)
output21s.txt	781 KB (800,000 bytes)	371 bytes (371 bytes)	1.88 KB (1,931 bytes)
output22s.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	220 bytes (220 bytes)
output23s.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	220 bytes (220 bytes)
output24s.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	215 bytes (215 bytes)
output25s.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	219 bytes (219 bytes)
output26s.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	218 bytes (218 bytes)
output27s.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	219 bytes (219 bytes)
output28s.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	220 bytes (220 bytes)

output29s.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	220 bytes (220 bytes)
output30s.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	220 bytes (220 bytes)
output31s.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	220 bytes (220 bytes)
output32s.txt	6.25 KB (6,400 bytes)	223 bytes (223 bytes)	219 bytes (219 bytes)
output33s.txt	6.25 KB (6,400 bytes)	223 bytes (223 bytes)	218 bytes (218 bytes)
output34s.txt	6.25 KB (6,400 bytes)	223 bytes (223 bytes)	218 bytes (218 bytes)
output35s.txt	6.25 KB (6,400 bytes)	223 bytes (223 bytes)	214 bytes (214 bytes)
output36s.txt	6.25 KB (6,400 bytes)	223 bytes (223 bytes)	217 bytes (217 bytes)
output37s.txt	6.25 KB (6,400 bytes)	223 bytes (223 bytes)	218 bytes (218 bytes)
output38s.txt	6.25 KB (6,400 bytes)	223 bytes (223 bytes)	219 bytes (219 bytes)
output39s.txt	6.25 KB (6,400 bytes)	223 bytes (223 bytes)	216 bytes (216 bytes)
output40s.txt	6.25 KB (6,400 bytes)	223 bytes (223 bytes)	218 bytes (218 bytes)
output41s.txt	78.1 KB (80,000 bytes)	267 bytes (267 bytes)	373 bytes (373 bytes)
output42s.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	216 bytes (216 bytes)
output43s.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	220 bytes (220 bytes)
output44s.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	218 bytes (218 bytes)
output45s.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	220 bytes (220 bytes)
output46s.txt	76.2 MB (80,000,000 bytes)	11.2 KB (11,543 bytes)	168 KB (172,900 bytes)

output47s.txt	78.1 KB (80,000 bytes)	267 bytes (267 bytes)	371 bytes (371 bytes)
output48s.txt	78.1 KB (80,000 bytes)	267 bytes (267 bytes)	371 bytes (371 bytes)
output49s.txt	78.1 KB (80,000 bytes)	267 bytes (267 bytes)	372 bytes (372 bytes)
output50s.txt	78.1 KB (80,000 bytes)	267 bytes (267 bytes)	372 bytes (372 bytes)

Tabla 55: Archivos encriptados con el algoritmo modificado AES comprimidos con la clave aaaaaaaaaaaaaaaa (mejor caso)

Elaboración: Propia

Clave: 1111111111111111			
Nombre del archivo	**Tamaño del archivo encriptado**	**Tamaño de archivo comprimido (LZMA)**	**Tamaño de archivo comprimido (deflate)**
output1z.txt	38.1 MB (40,000,000 bytes)	5.76 KB (5,899 bytes)	84.5 KB (86,550 bytes)
output2z.txt	78.1 KB (80,000 bytes)	265 bytes (265 bytes)	370 bytes (370 bytes)
output3z.txt	78.1 KB (80,000 bytes)	265 bytes (265 bytes)	370 bytes (370 bytes)
output4z.txt	78.1 KB (80,000 bytes)	265 bytes (265 bytes)	369 bytes (369 bytes)
output5z.txt	78.1 KB (80,000 bytes)	265 bytes (265 bytes)	371 bytes (371 bytes)
output6z.txt	78.1 KB (80,000 bytes)	265 bytes (265 bytes)	368 bytes (368 bytes)
output7z.txt	78.1 KB (80,000 bytes)	265 bytes (265 bytes)	370 bytes (370 bytes)
output8z.txt	78.1 KB (80,000 bytes)	265 bytes (265 bytes)	371 bytes (371 bytes)
output9z.txt	78.1 KB (80,000 bytes)	265 bytes (265 bytes)	370 bytes (370 bytes)
output10z.txt	78.1 KB (80,000 bytes)	267 bytes (267 bytes)	369 bytes (369 bytes)
output11z.txt	78.1 KB (80,000 bytes)	267 bytes (267 bytes)	373 bytes (373 bytes)

output12z.txt	781 KB (800,000 bytes)	371 bytes (371 bytes)	1.88 KB (1,930 bytes)
output13z.txt	781 KB (800,000 bytes)	371 bytes (371 bytes)	1.88 KB (1,927 bytes)
output14z.txt	781 KB (800,000 bytes)	371 bytes (371 bytes)	1.88 KB (1,927 bytes)
output15z.txt	781 KB (800,000 bytes)	371 bytes (371 bytes)	1.88 KB (1,931 bytes)
output16z.txt	781 KB (800,000 bytes)	371 bytes (371 bytes)	1.88 KB (1,930 bytes)
output17z.txt	781 KB (800,000 bytes)	371 bytes (371 bytes)	1.88 KB (1,931 bytes)
output18z.txt	781 KB (800,000 bytes)	371 bytes (371 bytes)	1.88 KB (1,929 bytes)
output19z.txt	781 KB (800,000 bytes)	371 bytes (371 bytes)	1.88 KB (1,929 bytes)
output20z.txt	781 KB (800,000 bytes)	371 bytes (371 bytes)	1.88 KB (1,927 bytes)
output21z.txt	781 KB (800,000 bytes)	371 bytes (371 bytes)	1.88 KB (1,929 bytes)
output22z.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	218 bytes (218 bytes)
output23z.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	220 bytes (220 bytes)
output24z.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	217 bytes (217 bytes)
output25z.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	220 bytes (220 bytes)
output26z.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	216 bytes (216 bytes)
output27z.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	219 bytes (219 bytes)
output28z.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	219 bytes (219 bytes)
output29z.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	221 bytes (221 bytes)
output30z.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	220 bytes (220 bytes)
output31z.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	220 bytes (220 bytes)
output32z.txt	6.25 KB (6,400 bytes)	223 bytes (223 bytes)	214 bytes (214 bytes)
output33z.txt	6.25 KB (6,400 bytes)	223 bytes (223 bytes)	219 bytes (219 bytes)

output34z.txt	6.25 KB (6,400 bytes)	223 bytes (223 bytes)	218 bytes (218 bytes)
output35z.txt	6.25 KB (6,400 bytes)	223 bytes (223 bytes)	214 bytes (214 bytes)
output36z.txt	6.25 KB (6,400 bytes)	223 bytes (223 bytes)	217 bytes (217 bytes)
output37z.txt	6.25 KB (6,400 bytes)	223 bytes (223 bytes)	214 bytes (214 bytes)
output38z.txt	6.25 KB (6,400 bytes)	223 bytes (223 bytes)	216 bytes (216 bytes)
output39z.txt	6.25 KB (6,400 bytes)	223 bytes (223 bytes)	217 bytes (217 bytes)
output40z.txt	6.25 KB (6,400 bytes)	223 bytes (223 bytes)	215 bytes (215 bytes)
output41z.txt	78.1 KB (80,000 bytes)	267 bytes (267 bytes)	374 bytes (374 bytes)
output42z.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	219 bytes (219 bytes)
output43z.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	221 bytes (221 bytes)
output44z.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	219 bytes (219 bytes)
output45z.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	220 bytes (220 bytes)
output46z.txt	76.2 MB (80,000,000 bytes)	11.2 KB (11,543 bytes)	168 KB (172,901 bytes)
output47z.txt	78.1 KB (80,000 bytes)	267 bytes (267 bytes)	372 bytes (372 bytes)
output48z.txt	78.1 KB (80,000 bytes)	267 bytes (267 bytes)	372 bytes (372 bytes)
output49z.txt	78.1 KB (80,000 bytes)	267 bytes (267 bytes)	373 bytes (373 bytes)
output50z.txt	78.1 KB (80,000 bytes)	267 bytes (267 bytes)	372 bytes (372 bytes)

Tabla 56: Archivos encriptados con el algoritmo modificado AES comprimidos con la clave aaaaaaaaaaaaaaaa (mejor caso)

Elaboracion: Propia

En este caso existe una gran disminución del tamaño de los archivos ya que la modificación del algoritmo Rijndael también es un algoritmo de

encriptación por bloques de 128 bits, debido a que el archivo a encriptar tiene caracteres repetitivos, el primer bloque comprimido se repetirá hasta cubrir la totalidad del archivo, al comprimir el archivo encriptado el resultado será igual a comprimir los primeros 128 bits multiplicado por el número de repeticiones de este hasta cubrir el tamaño total del archivo, sumando el tamaño los headers o cabeceras del algoritmo de compresión.

4.4. ANÁLISIS Y COMPARACIÓN DE LAS NUEVAS FUNCIONES CON LAS FUNCIONES ORIGINALES.

En esta sección compararemos el análisis de la complejidad temporal de ambos algoritmos, los tamaños de archivo comprimido del mejor y peor caso para medir el impacto que tiene la modificación en la entropía de caracteres una vez encriptado algún archivo.

4.4.1. ENTROPÍA DE LAS SALIDAS DEL ALGORITMO ORIGINAL CONTRA LA MODIFICACIÓN DEL ALGORITMO RIJNDAEL

Se comparó los tamaños de los archivos obtenidos al comprimir los archivos encriptados por el algoritmo original Rinjdael y la modificación de este, ya que estos tamaños dependen de encontrar una razón matemática entre los caracteres del archivo, la diferencia de tamaños nos dará una medida clara de cuál es el cambio en la entropía al modificar el algoritmo de encriptación Rijndael.

Se tomaron dos tipos de textos y claves, tanto para el mejor caso donde los caracteres son repetitivos, así como las claves; como para el peor caso, donde los caracteres son generados completamente aleatorios, de igual manera

las claves fueron escogidas con caracteres aleatorios, también se tomaron dos algoritmos de compresión diferentes, para tratar estos archivos.

4.4.1.1. PEOR CASO DEL ALGORITMO ORIGINAL VERSUS LA MODIFICACIÓN DEL ALGORITMO.

En los sub capítulos 4.1.2.1 y 4.3.3.1 se encripto y comprimió los mismos archivos con caracteres aleatorios utilizando la misma clave, en esta sección compararemos los tamaños de los archivos comprimidos para medir cual es el impacto de modificar el algoritmo Rijndael para el peor caso de entradas.

[redacted]COMPRIMIDO CON EL ALGORITMO LZMA

Al cruzar los datos obtenidos al comprimir los archivos encriptados con el algoritmo de compresión LZMA, para el peor caso se obtuvo los siguientes resultados.

Clave: 1Ds$f23-USAbCc12				
Nombre del archivo	**Tamaño del archivo**	**Tamaño de archivo comprimido**		**Dif.**
		Algoritmo AES original	**Algoritmo AES modificado**	
output1.txt	762 MB (800,000,000 bytes)	762 MB (800,000,156 bytes)	762 MB (800,000,156 bytes)	0
output2.txt	625 KB (640,000 bytes)	625 KB (640,156 bytes)	625 KB (640,156 bytes)	0
output3.txt	625 KB (640,000 bytes)	625 KB (640,156 bytes)	625 KB (640,156 bytes)	0
output4.txt	625 KB (640,000 bytes)	625 KB (640,156 bytes)	625 KB (640,156 bytes)	0
output5.txt	625 KB (640,000 bytes)	625 KB (640,156 bytes)	625 KB (640,156 bytes)	0

output6.txt	625 KB (640,000 bytes)	625 KB (640,156 bytes)	625 KB (640,156 bytes)	0
output7.txt	625 KB (640,000 bytes)	625 KB (640,156 bytes)	625 KB (640,156 bytes)	0
output8.txt	625 KB (640,000 bytes)	625 KB (640,156 bytes)	625 KB (640,156 bytes)	0
output9.txt	625 KB (640,000 bytes)	625 KB (640,156 bytes)	625 KB (640,156 bytes)	0
output10.txt	625 KB (640,000 bytes)	625 KB (640,158 bytes)	625 KB (640,158 bytes)	0
output11.txt	625 KB (640,000 bytes)	625 KB (640,158 bytes)	625 KB (640,158 bytes)	0
output12.txt	1.98 MB (2,080,000 bytes)	1.98 MB (2,080,158 bytes)	1.98 MB (2,080,158 bytes)	0
output13.txt	1.98 MB (2,080,000 bytes)	1.98 MB (2,080,158 bytes)	1.98 MB (2,080,158 bytes)	0
output14.txt	1.98 MB (2,080,000 bytes)	1.98 MB (2,080,158 bytes)	1.98 MB (2,080,158 bytes)	0
output15.txt	1.98 MB (2,080,000 bytes)	1.98 MB (2,080,158 bytes)	1.98 MB (2,080,158 bytes)	0
output16.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,158 bytes)	3.05 MB (3,200,158 bytes)	0
output17.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,158 bytes)	3.05 MB (3,200,158 bytes)	0
output18.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,158 bytes)	3.05 MB (3,200,158 bytes)	0
output19.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,158 bytes)	3.05 MB (3,200,158 bytes)	0
output20.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,158 bytes)	3.05 MB (3,200,158 bytes)	0
output21.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,158 bytes)	3.05 MB (3,200,158 bytes)	0
output22.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,158 bytes)	3.05 MB (3,200,158 bytes)	0
output23.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,158 bytes)	3.05 MB (3,200,158 bytes)	0
output24.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,158 bytes)	3.05 MB (3,200,158 bytes)	0
output25.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,158 bytes)	3.05 MB (3,200,158 bytes)	0

output26.txt	305 MB (320,000,000 bytes)	305 MB (320,000,158 bytes)	305 MB (320,000,158 bytes)	0
output27.txt	305 MB (320,000,000 bytes)	305 MB (320,000,158 bytes)	305 MB (320,000,158 bytes)	0
output28.txt	305 MB (320,000,000 bytes)	305 MB (320,000,158 bytes)	305 MB (320,000,158 bytes)	0
output29.txt	305 MB (320,000,000 bytes)	305 MB (320,000,158 bytes)	305 MB (320,000,158 bytes)	0
output30.txt	305 MB (320,000,000 bytes)	305 MB (320,000,158 bytes)	305 MB (320,000,158 bytes)	0
output31.txt	457 MB (480,000,000 bytes)	457 MB (480,000,158 bytes)	457 MB (480,000,158 bytes)	0
output32.txt	457 MB (480,000,000 bytes)	457 MB (480,000,158 bytes)	457 MB (480,000,158 bytes)	0
output33.txt	457 MB (480,000,000 bytes)	457 MB (480,000,158 bytes)	457 MB (480,000,158 bytes)	0
output34.txt	457 MB (480,000,000 bytes)	457 MB (480,000,158 bytes)	457 MB (480,000,158 bytes)	0
output35.txt	457 MB (480,000,000 bytes)	457 MB (480,000,158 bytes)	457 MB (480,000,158 bytes)	0
output36.txt	457 MB (480,000,000 bytes)	457 MB (480,000,158 bytes)	457 MB (480,000,158 bytes)	0
output37.txt	457 MB (480,000,000 bytes)	457 MB (480,000,158 bytes)	457 MB (480,000,158 bytes)	0
output38.txt	457 MB (480,000,000 bytes)	457 MB (480,000,158 bytes)	457 MB (480,000,158 bytes)	0
output39.txt	457 MB (480,000,000 bytes)	457 MB (480,000,158 bytes)	457 MB (480,000,158 bytes)	0
output40.txt	457 MB (480,000,000 bytes)	457 MB (480,000,158 bytes)	457 MB (480,000,158 bytes)	0
output41.txt	7.81 KB (8,000 bytes)	7.96 KB (8,158 bytes)	7.96 KB (8,158 bytes)	0
output42.txt	7.81 KB (8,000 bytes)	7.96 KB (8,158 bytes)	7.96 KB (8,158 bytes)	0
output43.txt	7.81 KB (8,000 bytes)	7.96 KB (8,158 bytes)	7.96 KB (8,158 bytes)	0
output44.txt	7.81 KB (8,000 bytes)	7.96 KB (8,158 bytes)	7.96 KB (8,158 bytes)	0
output45.txt	7.81 KB (8,000 bytes)	7.96 KB (8,158 bytes)	7.96 KB (8,158 bytes)	0

output46.txt	76.2 MB (80,000,000 bytes)	76.2 MB (80,000,158 bytes)	76.2 MB (80,000,158 bytes)	0
output47.txt	76.2 MB (80,000,000 bytes)	76.2 MB (80,000,158 bytes)	76.2 MB (80,000,158 bytes)	0
output48.txt	76.2 MB (80,000,000 bytes)	76.2 MB (80,000,158 bytes)	76.2 MB (80,000,158 bytes)	0
output49.txt	76.2 MB (80,000,000 bytes)	76.2 MB (80,000,158 bytes)	76.2 MB (80,000,158 bytes)	0
output50.txt	76.2 MB (80,000,000 bytes)	76.2 MB (80,000,158 bytes)	76.2 MB (80,000,158 bytes)	0

Tabla 57: Comparativa de tamaños de archivo comprimido para el peor caso con la clave 1Ds$f23-USAbCc12

Elaboración: Propia

Clave: abcdefghijklmnop				
Nombre del archivo	**Tamaño del archivo**	**Tamaño de archivo comprimido**		**Dif.**
		Algoritmo original AES	**Algoritmo modificado AES**	
output1b.txt	762 MB (800,000,000 bytes)	762 MB (800,000,158 bytes)	762 MB (800,000,158 bytes)	0
output2b.txt	625 KB (640,000 bytes)	625 KB (640,158 bytes)	625 KB (640,158 bytes)	0
output3b.txt	625 KB (640,000 bytes)	625 KB (640,158 bytes)	625 KB (640,158 bytes)	0
output4b.txt	625 KB (640,000 bytes)	625 KB (640,158 bytes)	625 KB (640,158 bytes)	0
output5b.txt	625 KB (640,000 bytes)	625 KB (640,158 bytes)	625 KB (640,158 bytes)	0
output6b.txt	625 KB (640,000 bytes)	625 KB (640,158 bytes)	625 KB (640,158 bytes)	0
output7b.txt	625 KB (640,000 bytes)	625 KB (640,158 bytes)	625 KB (640,158 bytes)	0

output8b.txt	625 KB (640,000 bytes)	625 KB (640,158 bytes)	625 KB (640,158 bytes)	0
output9b.txt	625 KB (640,000 bytes)	625 KB (640,158 bytes)	625 KB (640,158 bytes)	0
output10b.txt	625 KB (640,000 bytes)	625 KB (640,158 bytes)	625 KB (640,158 bytes)	0
output11b.txt	625 KB (640,000 bytes)	625 KB (640,158 bytes)	625 KB (640,158 bytes)	0
output12b.txt	1.98 MB (2,080,000 bytes)	1.98 MB (2,080,160 bytes)	1.98 MB (2,080,160 bytes)	0
output13b.txt	1.98 MB (2,080,000 bytes)	1.98 MB (2,080,160 bytes)	1.98 MB (2,080,160 bytes)	0
output14b.txt	1.98 MB (2,080,000 bytes)	1.98 MB (2,080,160 bytes)	1.98 MB (2,080,160 bytes)	0
output15b.txt	1.98 MB (2,080,000 bytes)	1.98 MB (2,080,160 bytes)	1.98 MB (2,080,160 bytes)	0
output16b.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,160 bytes)	3.05 MB (3,200,160 bytes)	0
output17b.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,160 bytes)	3.05 MB (3,200,160 bytes)	0
output18b.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,160 bytes)	3.05 MB (3,200,160 bytes)	0
output19b.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,160 bytes)	3.05 MB (3,200,160 bytes)	0
output20b.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,160 bytes)	3.05 MB (3,200,160 bytes)	0
output21b.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,160 bytes)	3.05 MB (3,200,160 bytes)	0
output22b.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,160 bytes)	3.05 MB (3,200,160 bytes)	0
output23b.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,160 bytes)	3.05 MB (3,200,160 bytes)	0
output24b.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,160 bytes)	3.05 MB (3,200,160 bytes)	0
output25b.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,160 bytes)	3.05 MB (3,200,160 bytes)	0

output26b.txt	305 MB (320,000,000 bytes)	305 MB (320,000,160 bytes)	305 MB (320,000,160 bytes)	0
output27b.txt	305 MB (320,000,000 bytes)	305 MB (320,000,160 bytes)	305 MB (320,000,160 bytes)	0
output28b.txt	305 MB (320,000,000 bytes)	305 MB (320,000,160 bytes)	305 MB (320,000,160 bytes)	0
output29b.txt	305 MB (320,000,000 bytes)	305 MB (320,000,160 bytes)	305 MB (320,000,160 bytes)	0
output30b.txt	305 MB (320,000,000 bytes)	305 MB (320,000,160 bytes)	305 MB (320,000,160 bytes)	0
output31b.txt	457 MB (480,000,000 bytes)	457 MB (480,000,160 bytes)	457 MB (480,000,160 bytes)	0
output32b.txt	457 MB (480,000,000 bytes)	457 MB (480,000,160 bytes)	457 MB (480,000,160 bytes)	0
output33b.txt	457 MB (480,000,000 bytes)	457 MB (480,000,160 bytes)	457 MB (480,000,160 bytes)	0
output34b.txt	457 MB (480,000,000 bytes)	457 MB (480,000,160 bytes)	457 MB (480,000,160 bytes)	0
output35b.txt	457 MB (480,000,000 bytes)	457 MB (480,000,160 bytes)	457 MB (480,000,160 bytes)	0
output36b.txt	457 MB (480,000,000 bytes)	457 MB (480,000,160 bytes)	457 MB (480,000,160 bytes)	0
output37b.txt	457 MB (480,000,000 bytes)	457 MB (480,000,160 bytes)	457 MB (480,000,160 bytes)	0
output38b.txt	457 MB (480,000,000 bytes)	457 MB (480,000,160 bytes)	457 MB (480,000,160 bytes)	0
output39b.txt	457 MB (480,000,000 bytes)	457 MB (480,000,160 bytes)	457 MB (480,000,160 bytes)	0
output40b.txt	457 MB (480,000,000 bytes)	457 MB (480,000,160 bytes)	457 MB (480,000,160 bytes)	0
output41b.txt	7.81 KB (8,000 bytes)	7.96 KB (8,160 bytes)	7.96 KB (8,160 bytes)	0
output42b.txt	7.81 KB (8,000 bytes)	7.96 KB (8,160 bytes)	7.96 KB (8,160 bytes)	0
output43b.txt	7.81 KB (8,000 bytes)	7.96 KB (8,160 bytes)	7.96 KB (8,160 bytes)	0

output44b.txt	7.81 KB (8,000 bytes)	7.96 KB (8,160 bytes)	7.96 KB (8,160 bytes)	0
output45b.txt	7.81 KB (8,000 bytes)	7.96 KB (8,160 bytes)	7.96 KB (8,160 bytes)	0
output46b.txt	76.2 MB (80,000,000 bytes)	76.2 MB (80,000,160 bytes)	76.2 MB (80,000,160 bytes)	0
output47b.txt	76.2 MB (80,000,000 bytes)	76.2 MB (80,000,160 bytes)	76.2 MB (80,000,160 bytes)	0
output48b.txt	76.2 MB (80,000,000 bytes)	76.2 MB (80,000,160 bytes)	76.2 MB (80,000,160 bytes)	0
output49b.txt	76.2 MB (80,000,000 bytes)	76.2 MB (80,000,160 bytes)	76.2 MB (80,000,160 bytes)	0
output50b.txt	76.2 MB (80,000,000 bytes)	76.2 MB (80,000,160 bytes)	76.2 MB (80,000,160 bytes)	0

Tabla 58: Comparativa de tamaños de archivo comprimido para el peor caso con la clave abcdefghijklmnop

Elaboración: Propia

No se observa ninguna variación en el tamaño de los archivos comprimidos aplicando la modificación en el algoritmo AES para el peor caso, ya que al encriptar un archivo con el algoritmo AES, este trata los caracteres de manera que el archivo final tiene caracteres independientes matemáticamente uno del otro, por lo que al aplicar el algoritmo de compresión este no encontrara alguna relación matemática entre estos, y se agregaran cabeceras propias del algoritmo de compresión por lo que el tamaño de este aumentara.

[redacted]COMPRIMIDO CON EL ALGORITMO DEFLATE

Al cruzar los datos obtenidos al comprimir los archivos encriptados con el algoritmo de compresión deflate, para el peor caso se obtuvo los siguientes resultados.

Clave: 1Ds$f23-USAbCc12				
Nombre del archivo	**Tamaño del archivo**	**Tamaño de archivo comprimido**		**Dif.**
		Algoritmo original AES	**Algoritmo modificado AES**	
output1.txt	762 MB (800,000,000 bytes)	762 MB (800,000,156 bytes)	762 MB (800,000,156 bytes)	0
output2.txt	625 KB (640,000 bytes)	625 KB (640,156 bytes)	625 KB (640,156 bytes)	0
output3.txt	625 KB (640,000 bytes)	625 KB (640,156 bytes)	625 KB (640,156 bytes)	0
output4.txt	625 KB (640,000 bytes)	625 KB (640,156 bytes)	625 KB (640,156 bytes)	0
output5.txt	625 KB (640,000 bytes)	625 KB (640,156 bytes)	625 KB (640,156 bytes)	0
output6.txt	625 KB (640,000 bytes)	625 KB (640,156 bytes)	625 KB (640,156 bytes)	0
output7.txt	625 KB (640,000 bytes)	625 KB (640,156 bytes)	625 KB (640,156 bytes)	0
output8.txt	625 KB (640,000 bytes)	625 KB (640,156 bytes)	625 KB (640,156 bytes)	0
output9.txt	625 KB (640,000 bytes)	625 KB (640,156 bytes)	625 KB (640,156 bytes)	0
output10.txt	625 KB (640,000 bytes)	625 KB (640,158 bytes)	625 KB (640,158 bytes)	0
output11.txt	625 KB (640,000 bytes)	625 KB (640,158 bytes)	625 KB (640,158 bytes)	0
output12.txt	1.98 MB (2,080,000 bytes)	1.98 MB (2,080,158 bytes)	1.98 MB (2,080,158 bytes)	0
output13.txt	1.98 MB (2,080,000 bytes)	1.98 MB (2,080,158 bytes)	1.98 MB (2,080,158 bytes)	0
output14.txt	1.98 MB (2,080,000 bytes)	1.98 MB (2,080,158 bytes)	1.98 MB (2,080,158 bytes)	0
output15.txt	1.98 MB (2,080,000 bytes)	1.98 MB (2,080,158 bytes)	1.98 MB (2,080,158 bytes)	0
output16.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,158 bytes)	3.05 MB (3,200,158 bytes)	0

output17.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,158 bytes)	3.05 MB (3,200,158 bytes)	0
output18.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,158 bytes)	3.05 MB (3,200,158 bytes)	0
output19.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,158 bytes)	3.05 MB (3,200,158 bytes)	0
output20.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,158 bytes)	3.05 MB (3,200,158 bytes)	0
output21.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,158 bytes)	3.05 MB (3,200,158 bytes)	0
output22.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,158 bytes)	3.05 MB (3,200,158 bytes)	0
output23.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,158 bytes)	3.05 MB (3,200,158 bytes)	0
output24.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,158 bytes)	3.05 MB (3,200,158 bytes)	0
output25.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,158 bytes)	3.05 MB (3,200,158 bytes)	0
output26.txt	305 MB (320,000,000 bytes)	305 MB (320,000,158 bytes)	305 MB (320,000,158 bytes)	0
output27.txt	305 MB (320,000,000 bytes)	305 MB (320,000,158 bytes)	305 MB (320,000,158 bytes)	0
output28.txt	305 MB (320,000,000 bytes)	305 MB (320,000,158 bytes)	305 MB (320,000,158 bytes)	0
output29.txt	305 MB (320,000,000 bytes)	305 MB (320,000,158 bytes)	305 MB (320,000,158 bytes)	0
output30.txt	305 MB (320,000,000 bytes)	305 MB (320,000,158 bytes)	305 MB (320,000,158 bytes)	0
output31.txt	457 MB (480,000,000 bytes)	457 MB (480,000,158 bytes)	457 MB (480,000,158 bytes)	0
output32.txt	457 MB (480,000,000 bytes)	457 MB (480,000,158 bytes)	457 MB (480,000,158 bytes)	0
output33.txt	457 MB (480,000,000 bytes)	457 MB (480,000,158 bytes)	457 MB (480,000,158 bytes)	0
output34.txt	457 MB (480,000,000 bytes)	457 MB (480,000,158 bytes)	457 MB (480,000,158 bytes)	0
output35.txt	457 MB (480,000,000 bytes)	457 MB (480,000,158 bytes)	457 MB (480,000,158 bytes)	0
output36.txt	457 MB (480,000,000 bytes)	457 MB (480,000,158 bytes)	457 MB (480,000,158 bytes)	0

output37.txt	457 MB (480,000,000 bytes)	457 MB (480,000,158 bytes)	457 MB (480,000,158 bytes)	0
output38.txt	457 MB (480,000,000 bytes)	457 MB (480,000,158 bytes)	457 MB (480,000,158 bytes)	0
output39.txt	457 MB (480,000,000 bytes)	457 MB (480,000,158 bytes)	457 MB (480,000,158 bytes)	0
output40.txt	457 MB (480,000,000 bytes)	457 MB (480,000,158 bytes)	457 MB (480,000,158 bytes)	0
output41.txt	7.81 KB (8,000 bytes)	7.96 KB (8,158 bytes)	7.96 KB (8,158 bytes)	0
output42.txt	7.81 KB (8,000 bytes)	7.96 KB (8,158 bytes)	7.96 KB (8,158 bytes)	0
output43.txt	7.81 KB (8,000 bytes)	7.96 KB (8,158 bytes)	7.96 KB (8,158 bytes)	0
output44.txt	7.81 KB (8,000 bytes)	7.96 KB (8,158 bytes)	7.96 KB (8,158 bytes)	0
output45.txt	7.81 KB (8,000 bytes)	7.96 KB (8,158 bytes)	7.96 KB (8,158 bytes)	0
output46.txt	76.2 MB (80,000,000 bytes)	76.2 MB (80,000,158 bytes)	76.2 MB (80,000,158 bytes)	0
output47.txt	76.2 MB (80,000,000 bytes)	76.2 MB (80,000,158 bytes)	76.2 MB (80,000,158 bytes)	0
output48.txt	76.2 MB (80,000,000 bytes)	76.2 MB (80,000,158 bytes)	76.2 MB (80,000,158 bytes)	0
output49.txt	76.2 MB (80,000,000 bytes)	76.2 MB (80,000,158 bytes)	76.2 MB (80,000,158 bytes)	0
output50.txt	76.2 MB (80,000,000 bytes)	76.2 MB (80,000,158 bytes)	76.2 MB (80,000,158 bytes)	0

Tabla 59: Comparativa de tamaños de archivo comprimido para el peor caso con la clave 1Ds$f23-USAbCc12

Elaboración: Propia

Clave: abcdefghijklmnop				
Nombre del archivo	**Tamaño del archivo**	**Tamaño de archivo comprimido**		**Dif.**
		Algoritmo original AES	**Algoritmo modificado AES**	
output1b.txt	762 MB (800,000,000 bytes)	762 MB (800,000,158 bytes)	762 MB (800,000,158 bytes)	0
output2b.txt	625 KB (640,000 bytes)	625 KB (640,158 bytes)	625 KB (640,158 bytes)	0
output3b.txt	625 KB (640,000 bytes)	625 KB (640,158 bytes)	625 KB (640,158 bytes)	0
output4b.txt	625 KB (640,000 bytes)	625 KB (640,158 bytes)	625 KB (640,158 bytes)	0
output5b.txt	625 KB (640,000 bytes)	625 KB (640,158 bytes)	625 KB (640,158 bytes)	0
output6b.txt	625 KB (640,000 bytes)	625 KB (640,158 bytes)	625 KB (640,158 bytes)	0
output7b.txt	625 KB (640,000 bytes)	625 KB (640,158 bytes)	625 KB (640,158 bytes)	0
output8b.txt	625 KB (640,000 bytes)	625 KB (640,158 bytes)	625 KB (640,158 bytes)	0
output9b.txt	625 KB (640,000 bytes)	625 KB (640,158 bytes)	625 KB (640,158 bytes)	0
output10b.txt	625 KB (640,000 bytes)	625 KB (640,160 bytes)	625 KB (640,160 bytes)	0
output11b.txt	625 KB (640,000 bytes)	625 KB (640,160 bytes)	625 KB (640,160 bytes)	0
output12b.txt	1.98 MB (2,080,000 bytes)	1.98 MB (2,080,160 bytes)	1.98 MB (2,080,160 bytes)	0
output13b.txt	1.98 MB (2,080,000 bytes)	1.98 MB (2,080,160 bytes)	1.98 MB (2,080,160 bytes)	0
output14b.txt	1.98 MB (2,080,000 bytes)	1.98 MB (2,080,160 bytes)	1.98 MB (2,080,160 bytes)	0
output15b.txt	1.98 MB (2,080,000 bytes)	1.98 MB (2,080,160 bytes)	1.98 MB (2,080,160 bytes)	0
output16b.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,160 bytes)	3.05 MB (3,200,160 bytes)	0

output17b.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,160 bytes)	3.05 MB (3,200,160 bytes)	0
output18b.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,160 bytes)	3.05 MB (3,200,160 bytes)	0
output19b.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,160 bytes)	3.05 MB (3,200,160 bytes)	0
output20b.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,160 bytes)	3.05 MB (3,200,160 bytes)	0
output21b.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,160 bytes)	3.05 MB (3,200,160 bytes)	0
output22b.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,160 bytes)	3.05 MB (3,200,160 bytes)	0
output23b.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,160 bytes)	3.05 MB (3,200,160 bytes)	0
output24b.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,160 bytes)	3.05 MB (3,200,160 bytes)	0
output25b.txt	3.05 MB (3,200,000 bytes)	3.05 MB (3,200,160 bytes)	3.05 MB (3,200,160 bytes)	0
output26b.txt	305 MB (320,000,000 bytes)	305 MB (320,000,160 bytes)	305 MB (320,000,160 bytes)	0
output27b.txt	305 MB (320,000,000 bytes)	305 MB (320,000,160 bytes)	305 MB (320,000,160 bytes)	0
output28b.txt	305 MB (320,000,000 bytes)	305 MB (320,000,160 bytes)	305 MB (320,000,160 bytes)	0
output29b.txt	305 MB (320,000,000 bytes)	305 MB (320,000,160 bytes)	305 MB (320,000,160 bytes)	0
output30b.txt	305 MB (320,000,000 bytes)	305 MB (320,000,160 bytes)	305 MB (320,000,160 bytes)	0
output31b.txt	457 MB (480,000,000 bytes)	457 MB (480,000,160 bytes)	457 MB (480,000,160 bytes)	0
output32b.txt	457 MB (480,000,000 bytes)	457 MB (480,000,160 bytes)	457 MB (480,000,160 bytes)	0
output33b.txt	457 MB (480,000,000 bytes)	457 MB (480,000,160 bytes)	457 MB (480,000,160 bytes)	0
output34b.txt	457 MB (480,000,000 bytes)	457 MB (480,000,160 bytes)	457 MB (480,000,160 bytes)	0
output35b.txt	457 MB (480,000,000 bytes)	457 MB (480,000,160 bytes)	457 MB (480,000,160 bytes)	0
output36b.txt	457 MB (480,000,000 bytes)	457 MB (480,000,160 bytes)	457 MB (480,000,160 bytes)	0

output37b.txt	457 MB (480,000,000 bytes)	457 MB (480,000,160 bytes)	457 MB (480,000,160 bytes)	0
output38b.txt	457 MB (480,000,000 bytes)	457 MB (480,000,160 bytes)	457 MB (480,000,160 bytes)	0
output39b.txt	457 MB (480,000,000 bytes)	457 MB (480,000,160 bytes)	457 MB (480,000,160 bytes)	0
output40b.txt	457 MB (480,000,000 bytes)	457 MB (480,000,160 bytes)	457 MB (480,000,160 bytes)	0
output41b.txt	7.81 KB (8,000 bytes)	7.96 KB (8,160 bytes)	7.96 KB (8,160 bytes)	0
output42b.txt	7.81 KB (8,000 bytes)	7.96 KB (8,160 bytes)	7.96 KB (8,160 bytes)	0
output43b.txt	7.81 KB (8,000 bytes)	7.96 KB (8,160 bytes)	7.96 KB (8,160 bytes)	0
output44b.txt	7.81 KB (8,000 bytes)	7.96 KB (8,160 bytes)	7.96 KB (8,160 bytes)	0
output45b.txt	7.81 KB (8,000 bytes)	7.96 KB (8,160 bytes)	7.96 KB (8,160 bytes)	0
output46b.txt	76.2 MB (80,000,000 bytes)	76.2 MB (80,000,160 bytes)	76.2 MB (80,000,160 bytes)	0
output47b.txt	76.2 MB (80,000,000 bytes)	76.2 MB (80,000,160 bytes)	76.2 MB (80,000,160 bytes)	0
output48b.txt	76.2 MB (80,000,000 bytes)	76.2 MB (80,000,160 bytes)	76.2 MB (80,000,160 bytes)	0
output49b.txt	76.2 MB (80,000,000 bytes)	76.2 MB (80,000,160 bytes)	76.2 MB (80,000,160 bytes)	0
output50b.txt	76.2 MB (80,000,000 bytes)	76.2 MB (80,000,160 bytes)	76.2 MB (80,000,160 bytes)	0

Tabla 60: Comparativa de tamaños de archivo comprimido para el peor caso con la clave abcdefghijklmnop

Elaboración: Propia

No se encuentra diferencias significativas para el peor caso ya que la dependencia matemática entre los caracteres de los archivos es mínima, tanto para el algoritmo original como para la modificación propuesta.

4.4.1.2. MEJOR CASO DEL ALGORITMO ORIGINAL VERSUS LA MODIFICACIÓN DEL ALGORITMO

En los sub capítulos 4.1.2.2 y 4.3.3.2 se encripto y comprimió los mismos archivos con caracteres repetitivos de diferentes tamaños utilizando la misma clave de caracteres también repetitivos, con estas entradas los archivos generados por el algoritmo de encriptación serán los suficientemente simples y repetitivos para aplicar el algoritmo de compresión, de esta manera en esta sección compararemos los tamaños de los archivos comprimidos para medir cual es el impacto de modificar el algoritmo Rijndael para el mejor caso de entradas y clave de cifrado.

[redacted]COMPRIMIDO CON EL ALGORITMO LZMA

Al cruzar los datos obtenidos al comprimir los archivos encriptados con el algoritmo de compresión LZMA, para el peor caso se obtuvo los siguientes resultados.

Clave: aaaaaaaaaaaaaaaa				
Nombre del archivo	**Tamaño del archivo**	**Tamaño de archivo comprimido**		**Dif.**
		Algoritmo AES original	**Algoritmo AES modificado**	
output1s.txt	762 MB (800,000,000 bytes)	5.76 KB (5,899 bytes)	5.76 KB (5,899 bytes)	0
output2s.txt	625 KB (640,000 bytes)	265 bytes (265 bytes)	265 bytes (265 bytes)	0
output3s.txt	625 KB (640,000 bytes)	265 bytes (265 bytes)	265 bytes (265 bytes)	0
output4s.txt	625 KB (640,000 bytes)	265 bytes (265 bytes)	265 bytes (265 bytes)	0

output5s.txt	625 KB (640,000 bytes)	265 bytes (265 bytes)	265 bytes (265 bytes)	0
output6s.txt	625 KB (640,000 bytes)	265 bytes (265 bytes)	265 bytes (265 bytes)	0
output7s.txt	625 KB (640,000 bytes)	265 bytes (265 bytes)	265 bytes (265 bytes)	0
output8s.txt	625 KB (640,000 bytes)	265 bytes (265 bytes)	265 bytes (265 bytes)	0
output9s.txt	625 KB (640,000 bytes)	265 bytes (265 bytes)	265 bytes (265 bytes)	0
output10s.txt	625 KB (640,000 bytes)	265 bytes (265 bytes)	267 bytes (267 bytes)	2
output11s.txt	625 KB (640,000 bytes)	267 bytes (267 bytes)	267 bytes (267 bytes)	0
output12s.txt	1.98 MB (2,080,000 bytes)	370 bytes (370 bytes)	371 bytes (371 bytes)	1
output13s.txt	1.98 MB (2,080,000 bytes)	371 bytes (371 bytes)	371 bytes (371 bytes)	0
output14s.txt	1.98 MB (2,080,000 bytes)	370 bytes (370 bytes)	371 bytes (371 bytes)	1
output15s.txt	1.98 MB (2,080,000 bytes)	371 bytes (371 bytes)	371 bytes (371 bytes)	0
output16s.txt	3.05 MB (3,200,000 bytes)	371 bytes (371 bytes)	371 bytes (371 bytes)	0
output17s.txt	3.05 MB (3,200,000 bytes)	371 bytes (371 bytes)	371 bytes (371 bytes)	0
output18s.txt	3.05 MB (3,200,000 bytes)	371 bytes (371 bytes)	371 bytes (371 bytes)	0
output19s.txt	3.05 MB (3,200,000 bytes)	371 bytes (371 bytes)	371 bytes (371 bytes)	0
output20s.txt	3.05 MB (3,200,000 bytes)	371 bytes (371 bytes)	371 bytes (371 bytes)	0
output21s.txt	3.05 MB (3,200,000 bytes)	371 bytes (371 bytes)	371 bytes (371 bytes)	0
output22s.txt	3.05 MB (3,200,000 bytes)	227 bytes (227 bytes)	227 bytes (227 bytes)	0
output23s.txt	3.05 MB (3,200,000 bytes)	227 bytes (227 bytes)	227 bytes (227 bytes)	0
output24s.txt	3.05 MB (3,200,000 bytes)	227 bytes (227 bytes)	227 bytes (227 bytes)	0

output25s.txt	3.05 MB (3,200,000 bytes)	227 bytes (227 bytes)	227 bytes (227 bytes)	0
output26s.txt	305 MB (320,000,000 bytes)	227 bytes (227 bytes)	227 bytes (227 bytes)	0
output27s.txt	305 MB (320,000,000 bytes)	227 bytes (227 bytes)	227 bytes (227 bytes)	0
output28s.txt	305 MB (320,000,000 bytes)	227 bytes (227 bytes)	227 bytes (227 bytes)	0
output29s.txt	305 MB (320,000,000 bytes)	227 bytes (227 bytes)	227 bytes (227 bytes)	0
output30s.txt	305 MB (320,000,000 bytes)	227 bytes (227 bytes)	227 bytes (227 bytes)	0
output31s.txt	457 MB (480,000,000 bytes)	227 bytes (227 bytes)	227 bytes (227 bytes)	0
output32s.txt	457 MB (480,000,000 bytes)	223 bytes (223 bytes)	223 bytes (223 bytes)	0
output33s.txt	457 MB (480,000,000 bytes)	223 bytes (223 bytes)	223 bytes (223 bytes)	0
output34s.txt	457 MB (480,000,000 bytes)	223 bytes (223 bytes)	223 bytes (223 bytes)	0
output35s.txt	457 MB (480,000,000 bytes)	223 bytes (223 bytes)	223 bytes (223 bytes)	0
output36s.txt	457 MB (480,000,000 bytes)	223 bytes (223 bytes)	223 bytes (223 bytes)	0
output37s.txt	457 MB (480,000,000 bytes)	223 bytes (223 bytes)	223 bytes (223 bytes)	0
output38s.txt	457 MB (480,000,000 bytes)	223 bytes (223 bytes)	223 bytes (223 bytes)	0
output39s.txt	457 MB (480,000,000 bytes)	223 bytes (223 bytes)	223 bytes (223 bytes)	0
output40s.txt	457 MB (480,000,000 bytes)	223 bytes (223 bytes)	223 bytes (223 bytes)	0
output41s.txt	7.81 KB (8,000 bytes)	267 bytes (267 bytes)	267 bytes (267 bytes)	0
output42s.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	227 bytes (227 bytes)	0
output43s.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	227 bytes (227 bytes)	0
output44s.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	227 bytes (227 bytes)	0

output45s.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	227 bytes (227 bytes)	0
output46s.txt	76.2 MB (80,000,000 bytes)	11.2 KB (11,543 bytes)	11.2 KB (11,543 bytes)	0
output47s.txt	76.2 MB (80,000,000 bytes)	267 bytes (267 bytes)	267 bytes (267 bytes)	0
output48s.txt	76.2 MB (80,000,000 bytes)	267 bytes (267 bytes)	267 bytes (267 bytes)	0
output49s.txt	76.2 MB (80,000,000 bytes)	267 bytes (267 bytes)	267 bytes (267 bytes)	0
output50s.txt	76.2 MB (80,000,000 bytes)	267 bytes (267 bytes)	267 bytes (267 bytes)	0

Tabla 61: Comparativa de tamaños de archivo comprimido para el mejor caso con la clave aaaaaaaaaaaaaaa

Elaboración: Propia

Clave: 1111111111111111				
Nombre del archivo	**Tamaño del archivo**	**Tamaño de archivo comprimido**		**Dif.**
		Algoritmo AES original	**Algoritmo AES modificado**	
output1z.txt	762 MB (800,000,000 bytes)	5.76 KB (5,899 bytes)	5.76 KB (5,899 bytes)	0
output2z.txt	625 KB (640,000 bytes)	265 bytes (265 bytes)	265 bytes (265 bytes)	0
output3z.txt	625 KB (640,000 bytes)	265 bytes (265 bytes)	265 bytes (265 bytes)	0
output4z.txt	625 KB (640,000 bytes)	265 bytes (265 bytes)	265 bytes (265 bytes)	0
output5z.txt	625 KB (640,000 bytes)	265 bytes (265 bytes)	265 bytes (265 bytes)	0
output6z.txt	625 KB (640,000 bytes)	265 bytes (265 bytes)	265 bytes (265 bytes)	0
output7z.txt	625 KB (640,000 bytes)	265 bytes (265 bytes)	265 bytes (265 bytes)	0

output8z.txt	625 KB (640,000 bytes)	265 bytes (265 bytes)	265 bytes (265 bytes)	0
output9z.txt	625 KB (640,000 bytes)	265 bytes (265 bytes)	265 bytes (265 bytes)	0
output10z.txt	625 KB (640,000 bytes)	267 bytes (267 bytes)	267 bytes (267 bytes)	0
output11z.txt	625 KB (640,000 bytes)	267 bytes (267 bytes)	267 bytes (267 bytes)	0
output12z.txt	1.98 MB (2,080,000 bytes)	371 bytes (371 bytes)	371 bytes (371 bytes)	0
output13z.txt	1.98 MB (2,080,000 bytes)	371 bytes (371 bytes)	371 bytes (371 bytes)	0
output14z.txt	1.98 MB (2,080,000 bytes)	371 bytes (371 bytes)	371 bytes (371 bytes)	0
output15z.txt	1.98 MB (2,080,000 bytes)	371 bytes (371 bytes)	371 bytes (371 bytes)	0
output16z.txt	3.05 MB (3,200,000 bytes)	371 bytes (371 bytes)	371 bytes (371 bytes)	0
output17z.txt	3.05 MB (3,200,000 bytes)	371 bytes (371 bytes)	371 bytes (371 bytes)	0
output18z.txt	3.05 MB (3,200,000 bytes)	371 bytes (371 bytes)	371 bytes (371 bytes)	0
output19z.txt	3.05 MB (3,200,000 bytes)	371 bytes (371 bytes)	371 bytes (371 bytes)	0
output20z.txt	3.05 MB (3,200,000 bytes)	371 bytes (371 bytes)	371 bytes (371 bytes)	0
output21z.txt	3.05 MB (3,200,000 bytes)	371 bytes (371 bytes)	371 bytes (371 bytes)	0
output22z.txt	3.05 MB (3,200,000 bytes)	227 bytes (227 bytes)	227 bytes (227 bytes)	0
output23z.txt	3.05 MB (3,200,000 bytes)	227 bytes (227 bytes)	227 bytes (227 bytes)	0
output24z.txt	3.05 MB (3,200,000 bytes)	227 bytes (227 bytes)	227 bytes (227 bytes)	0
output25z.txt	3.05 MB (3,200,000 bytes)	227 bytes (227 bytes)	227 bytes (227 bytes)	0
output26z.txt	305 MB (320,000,000 bytes)	227 bytes (227 bytes)	227 bytes (227 bytes)	0
output27z.txt	305 MB (320,000,000 bytes)	227 bytes (227 bytes)	227 bytes (227 bytes)	0

output28z.txt	305 MB (320,000,000 bytes)	227 bytes (227 bytes)	227 bytes (227 bytes)	0
output29z.txt	305 MB (320,000,000 bytes)	227 bytes (227 bytes)	227 bytes (227 bytes)	0
output30z.txt	305 MB (320,000,000 bytes)	227 bytes (227 bytes)	227 bytes (227 bytes)	0
output31z.txt	457 MB (480,000,000 bytes)	227 bytes (227 bytes)	227 bytes (227 bytes)	0
output32z.txt	457 MB (480,000,000 bytes)	223 bytes (223 bytes)	223 bytes (223 bytes)	0
output33z.txt	457 MB (480,000,000 bytes)	223 bytes (223 bytes)	223 bytes (223 bytes)	0
output34z.txt	457 MB (480,000,000 bytes)	223 bytes (223 bytes)	223 bytes (223 bytes)	0
output35z.txt	457 MB (480,000,000 bytes)	223 bytes (223 bytes)	223 bytes (223 bytes)	0
output36z.txt	457 MB (480,000,000 bytes)	223 bytes (223 bytes)	223 bytes (223 bytes)	0
output37z.txt	457 MB (480,000,000 bytes)	223 bytes (223 bytes)	223 bytes (223 bytes)	0
output38z.txt	457 MB (480,000,000 bytes)	223 bytes (223 bytes)	223 bytes (223 bytes)	0
output39z.txt	457 MB (480,000,000 bytes)	223 bytes (223 bytes)	223 bytes (223 bytes)	0
output40z.txt	457 MB (480,000,000 bytes)	223 bytes (223 bytes)	223 bytes (223 bytes)	0
output41z.txt	7.81 KB (8,000 bytes)	267 bytes (267 bytes)	267 bytes (267 bytes)	0
output42z.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	227 bytes (227 bytes)	0
output43z.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	227 bytes (227 bytes)	0
output44z.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	227 bytes (227 bytes)	0
output45z.txt	7.81 KB (8,000 bytes)	227 bytes (227 bytes)	227 bytes (227 bytes)	0
output46z.txt	76.2 MB (80,000,000 bytes)	11.2 KB (11,543 bytes)	11.2 KB (11,543 bytes)	0
output47z.txt	76.2 MB (80,000,000 bytes)	267 bytes (267 bytes)	267 bytes (267 bytes)	0

output48z.txt	76.2 MB (80,000,000 bytes)	267 bytes (267 bytes)	267 bytes (267 bytes)	0
output49z.txt	76.2 MB (80,000,000 bytes)	267 bytes (267 bytes)	267 bytes (267 bytes)	0
output50z.txt	76.2 MB (80,000,000 bytes)	267 bytes (267 bytes)	267 bytes (267 bytes)	0

Tabla 62: Comparativa de tamaños de archivo comprimido para el mejor caso con la clave 1111111111111111

Elaboración: Propia

Con el algoritmo LZMA notamos una pequeña diferencia a favor de la modificación del algoritmo Rijndael, pero estos datos no son suficientes para admitir o negar la hipótesis.

Con el algoritmo de comprensión DEFLATE se obtuvieron diferencias en los tamaños de archivo más notables, estos resultados serán expuestos en el sub capitulo siguiente.

[redacted]COMPRIMIDO CON EL ALGORITMO DEFLATE

Al cruzar los datos obtenidos al comprimir los archivos encriptados con el algoritmo de compresión deflate, para el mejor caso se obtuvo los siguientes resultados.

Clave: aaaaaaaaaaaaaaaa				
Nombre del archivo	**Tamaño del archivo**	**Tamaño de archivo comprimido**		**Dif.**
		Algoritmo AES original	**Algoritmo AES modificado**	
output1s.txt	762 MB (800,000,000 bytes)	84.5 KB (86,549 bytes)	84.5 KB (86,550 bytes)	-1
output2s.txt	625 KB (640,000 bytes)	371 bytes (371 bytes)	370 bytes (370 bytes)	-1
output3s.txt	625 KB (640,000 bytes)	367 bytes (367 bytes)	369 bytes (369 bytes)	2
output4s.txt	625 KB (640,000 bytes)	369 bytes (369 bytes)	372 bytes (372 bytes)	3
output5s.txt	625 KB (640,000 bytes)	370 bytes (370 bytes)	368 bytes (368 bytes)	-2
output6s.txt	625 KB (640,000 bytes)	372 bytes (372 bytes)	372 bytes (372 bytes)	0
output7s.txt	625 KB (640,000 bytes)	368 bytes (368 bytes)	370 bytes (370 bytes)	2
output8s.txt	625 KB (640,000 bytes)	370 bytes (370 bytes)	371 bytes (371 bytes)	1
output9s.txt	625 KB (640,000 bytes)	367 bytes (367 bytes)	369 bytes (369 bytes)	2
output10s.txt	625 KB (640,000 bytes)	370 bytes (370 bytes)	369 bytes (369 bytes)	-1
output11s.txt	625 KB (640,000 bytes)	372 bytes (372 bytes)	372 bytes (372 bytes)	0
output12s.txt	1.98 MB (2,080,000 bytes)	1.88 KB (1,928 bytes)	1.88 KB (1,926 bytes)	-2
output13s.txt	1.98 MB (2,080,000 bytes)	1.88 KB (1,930 bytes)	1.88 KB (1,928 bytes)	0
output14s.txt	1.98 MB (2,080,000 bytes)	1.88 KB (1,928 bytes)	1.88 KB (1,927 bytes)	-1
output15s.txt	1.98 MB (2,080,000 bytes)	1.88 KB (1,930 bytes)	1.88 KB (1,930 bytes)	0
output16s.txt	3.05 MB (3,200,000 bytes)	1.88 KB (1,928 bytes)	1.88 KB (1,930 bytes)	2

output17s.txt	3.05 MB (3,200,000 bytes)	1.88 KB (1,930 bytes)	1.88 KB (1,931 bytes)	1
output18s.txt	3.05 MB (3,200,000 bytes)	1.88 KB (1,929 bytes)	1.88 KB (1,932 bytes)	3
output19s.txt	3.05 MB (3,200,000 bytes)	1.88 KB (1,927 bytes)	1.88 KB (1,931 bytes)	4
output20s.txt	3.05 MB (3,200,000 bytes)	1.88 KB (1,929 bytes)	1.88 KB (1,929 bytes)	0
output21s.txt	3.05 MB (3,200,000 bytes)	1.88 KB (1,927 bytes)	1.88 KB (1,931 bytes)	4
output22s.txt	3.05 MB (3,200,000 bytes)	219 bytes (219 bytes)	220 bytes (220 bytes)	1
output23s.txt	3.05 MB (3,200,000 bytes)	219 bytes (219 bytes)	220 bytes (220 bytes)	1
output24s.txt	3.05 MB (3,200,000 bytes)	217 bytes (217 bytes)	215 bytes (215 bytes)	-2
output25s.txt	3.05 MB (3,200,000 bytes)	220 bytes (220 bytes)	219 bytes (219 bytes)	-1
output26s.txt	305 MB (320,000,000 bytes)	220 bytes (220 bytes)	218 bytes (218 bytes)	-2
output27s.txt	305 MB (320,000,000 bytes)	218 bytes (218 bytes)	219 bytes (219 bytes)	1
output28s.txt	305 MB (320,000,000 bytes)	220 bytes (220 bytes)	220 bytes (220 bytes)	0
output29s.txt	305 MB (320,000,000 bytes)	220 bytes (220 bytes)	220 bytes (220 bytes)	0
output30s.txt	305 MB (320,000,000 bytes)	219 bytes (219 bytes)	220 bytes (220 bytes)	1
output31s.txt	457 MB (480,000,000 bytes)	219 bytes (219 bytes)	220 bytes (220 bytes)	1
output32s.txt	457 MB (480,000,000 bytes)	218 bytes (218 bytes)	219 bytes (219 bytes)	1
output33s.txt	457 MB (480,000,000 bytes)	217 bytes (217 bytes)	218 bytes (218 bytes)	1
output34s.txt	457 MB (480,000,000 bytes)	217 bytes (217 bytes)	218 bytes (218 bytes)	1
output35s.txt	457 MB (480,000,000 bytes)	215 bytes (215 bytes)	214 bytes (214 bytes)	-1
output36s.txt	457 MB (480,000,000 bytes)	216 bytes (216 bytes)	217 bytes (217 bytes)	1

output37s.txt	457 MB (480,000,000 bytes)	217 bytes (217 bytes)	218 bytes (218 bytes)	1
output38s.txt	457 MB (480,000,000 bytes)	219 bytes (219 bytes)	219 bytes (219 bytes)	0
output39s.txt	457 MB (480,000,000 bytes)	214 bytes (214 bytes)	216 bytes (216 bytes)	2
output40s.txt	457 MB (480,000,000 bytes)	214 bytes (214 bytes)	218 bytes (218 bytes)	4
output41s.txt	7.81 KB (8,000 bytes)	372 bytes (372 bytes)	373 bytes (373 bytes)	1
output42s.txt	7.81 KB (8,000 bytes)	218 bytes (218 bytes)	216 bytes (216 bytes)	-2
output43s.txt	7.81 KB (8,000 bytes)	219 bytes (219 bytes)	220 bytes (220 bytes)	1
output44s.txt	7.81 KB (8,000 bytes)	216 bytes (216 bytes)	218 bytes (218 bytes)	2
output45s.txt	7.81 KB (8,000 bytes)	219 bytes (219 bytes)	220 bytes (220 bytes)	1
output46s.txt	76.2 MB (80,000,000 bytes)	168 KB (172,900 bytes)	168 KB (172,900 bytes)	0
output47s.txt	76.2 MB (80,000,000 bytes)	371 bytes (371 bytes)	371 bytes (371 bytes)	0
output48s.txt	76.2 MB (80,000,000 bytes)	371 bytes (371 bytes)	371 bytes (371 bytes)	0
output49s.txt	76.2 MB (80,000,000 bytes)	373 bytes (373 bytes)	372 bytes (372 bytes)	-1
output50s.txt	76.2 MB (80,000,000 bytes)	370 bytes (370 bytes)	372 bytes (372 bytes)	2

Tabla 63: Comparativa de tamaños de archivo comprimido para el mejor caso con la clave aaaaaaaaaaaaaaaa

Elaboración: Propia

Clave: 1111111111111111				
Nombre del archivo	**Tamaño del archivo**	**Tamaño de archivo comprimido**		**Dif.**
		Algoritmo AES original	**Algoritmo AES modificado**	
output1z.txt	762 MB (800,000,000 bytes)	84.5 KB (86,550 bytes)	84.5 KB (86,550 bytes)	0
output2z.txt	625 KB (640,000 bytes)	370 bytes (370 bytes)	370 bytes (370 bytes)	0
output3z.txt	625 KB (640,000 bytes)	370 bytes (370 bytes)	370 bytes (370 bytes)	0
output4z.txt	625 KB (640,000 bytes)	369 bytes (369 bytes)	369 bytes (369 bytes)	0
output5z.txt	625 KB (640,000 bytes)	369 bytes (369 bytes)	371 bytes (371 bytes)	2
output6z.txt	625 KB (640,000 bytes)	369 bytes (369 bytes)	368 bytes (368 bytes)	-1
output7z.txt	625 KB (640,000 bytes)	370 bytes (370 bytes)	370 bytes (370 bytes)	0
output8z.txt	625 KB (640,000 bytes)	371 bytes (371 bytes)	371 bytes (371 bytes)	0
output9z.txt	625 KB (640,000 bytes)	370 bytes (370 bytes)	370 bytes (370 bytes)	0
output10z.txt	625 KB (640,000 bytes)	370 bytes (370 bytes)	369 bytes (369 bytes)	-1
output11z.txt	625 KB (640,000 bytes)	371 bytes (371 bytes)	373 bytes (373 bytes)	2
output12z.txt	1.98 MB (2,080,000 bytes)	1.88 KB (1,926 bytes)	1.88 KB (1,930 bytes)	4
output13z.txt	1.98 MB (2,080,000 bytes)	1.88 KB (1,929 bytes)	1.88 KB (1,927 bytes)	-2
output14z.txt	1.98 MB (2,080,000 bytes)	1.88 KB (1,928 bytes)	1.88 KB (1,927 bytes)	-1
output15z.txt	1.98 MB (2,080,000 bytes)	1.88 KB (1,929 bytes)	1.88 KB (1,931 bytes)	2
output16z.txt	3.05 MB (3,200,000 bytes)	1.88 KB (1,931 bytes)	1.88 KB (1,930 bytes)	-1

output17z.txt	3.05 MB (3,200,000 bytes)	1.88 KB (1,931 bytes)	1.88 KB (1,931 bytes)	0
output18z.txt	3.05 MB (3,200,000 bytes)	1.88 KB (1,929 bytes)	1.88 KB (1,929 bytes)	0
output19z.txt	3.05 MB (3,200,000 bytes)	1.88 KB (1,931 bytes)	1.88 KB (1,929 bytes)	-2
output20z.txt	3.05 MB (3,200,000 bytes)	1.88 KB (1,927 bytes)	1.88 KB (1,927 bytes)	0
output21z.txt	3.05 MB (3,200,000 bytes)	1.88 KB (1,931 bytes)	1.88 KB (1,929 bytes)	-2
output22z.txt	3.05 MB (3,200,000 bytes)	220 bytes (220 bytes)	218 bytes (218 bytes)	-2
output23z.txt	3.05 MB (3,200,000 bytes)	218 bytes (218 bytes)	220 bytes (220 bytes)	2
output24z.txt	3.05 MB (3,200,000 bytes)	217 bytes (217 bytes)	217 bytes (217 bytes)	0
output25z.txt	3.05 MB (3,200,000 bytes)	221 bytes (221 bytes)	220 bytes (220 bytes)	-1
output26z.txt	305 MB (320,000,000 bytes)	217 bytes (217 bytes)	216 bytes (216 bytes)	-1
output27z.txt	305 MB (320,000,000 bytes)	218 bytes (218 bytes)	219 bytes (219 bytes)	1
output28z.txt	305 MB (320,000,000 bytes)	220 bytes (220 bytes)	219 bytes (219 bytes)	-1
output29z.txt	305 MB (320,000,000 bytes)	219 bytes (219 bytes)	221 bytes (221 bytes)	2
output30z.txt	305 MB (320,000,000 bytes)	219 bytes (219 bytes)	220 bytes (220 bytes)	1
output31z.txt	457 MB (480,000,000 bytes)	219 bytes (219 bytes)	220 bytes (220 bytes)	1
output32z.txt	457 MB (480,000,000 bytes)	216 bytes (216 bytes)	214 bytes (214 bytes)	-2
output33z.txt	457 MB (480,000,000 bytes)	218 bytes (218 bytes)	219 bytes (219 bytes)	1
output34z.txt	457 MB (480,000,000 bytes)	216 bytes (216 bytes)	218 bytes (218 bytes)	2
output35z.txt	457 MB (480,000,000 bytes)	215 bytes (215 bytes)	214 bytes (214 bytes)	-1
output36z.txt	457 MB (480,000,000 bytes)	218 bytes (218 bytes)	217 bytes (217 bytes)	-1

output37z.txt	457 MB (480,000,000 bytes)	213 bytes (213 bytes)	214 bytes (214 bytes)	1
output38z.txt	457 MB (480,000,000 bytes)	218 bytes (218 bytes)	216 bytes (216 bytes)	-2
output39z.txt	457 MB (480,000,000 bytes)	217 bytes (217 bytes)	217 bytes (217 bytes)	0
output40z.txt	457 MB (480,000,000 bytes)	218 bytes (218 bytes)	215 bytes (215 bytes)	-3
output41z.txt	7.81 KB (8,000 bytes)	372 bytes (372 bytes)	374 bytes (374 bytes)	2
output42z.txt	7.81 KB (8,000 bytes)	214 bytes (214 bytes)	219 bytes (219 bytes)	5
output43z.txt	7.81 KB (8,000 bytes)	220 bytes (220 bytes)	221 bytes (221 bytes)	1
output44z.txt	7.81 KB (8,000 bytes)	219 bytes (219 bytes)	219 bytes (219 bytes)	0
output45z.txt	7.81 KB (8,000 bytes)	219 bytes (219 bytes)	220 bytes (220 bytes)	1
output46z.txt	76.2 MB (80,000,000 bytes)	168 KB (172,901 bytes)	168 KB (172,901 bytes)	0
output47z.txt	76.2 MB (80,000,000 bytes)	373 bytes (373 bytes)	372 bytes (372 bytes)	-1
output48z.txt	76.2 MB (80,000,000 bytes)	373 bytes (373 bytes)	372 bytes (372 bytes)	-1
output49z.txt	76.2 MB (80,000,000 bytes)	374 bytes (374 bytes)	373 bytes (373 bytes)	-1
output50z.txt	76.2 MB (80,000,000 bytes)	372 bytes (372 bytes)	372 bytes (372 bytes)	0

Tabla 64: Comparativa de tamaños de archivo comprimido para el mejor caso con la clave 1111111111111111

Elaboración: Propia

Para comprobar que la diferencia encontrada entre los dos algoritmos no es una diferencia al azar se utilizara la distribución normal o distribución de gauss de diferencia de medias entonces las hipótesis son:

$$H_o = \overline{tam_m} - \overline{tam_o} \geq 0$$

$$H_1 = \overline{tam_m} - \overline{tam_o} < 0$$

Aunque el estadístico que correspondería a este test es el asociado a una distribución T-Student, por ser las desviaciones de las poblaciones desconocidas, como el tamaño de las muestras es elevado y sabemos que una distribución T-Student con muchos grados de libertad se aproximaba mucho a una Normal, utilizaremos el siguiente estadístico de la **ecuación 2**:

$$Z = \frac{(\bar{X}_1 - \bar{X}_2) - (\mu_1 - \mu_2)}{\sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}}$$

Donde:

$\bar{X}_1 - \bar{X}_2$: Diferencia de medias de las muestras.

$\mu_1 - \mu_2$: Diferencias de medias poblacional.

σ_1 : Desviación estándar de la primera muestra.

σ_2 : Desviación estándar de la segunda muestra.

n_1 : Tamaño de la primera muestra.

n_2 : Tamaño de la segunda muestra.

Con un nivel de confiabilidad del 99%, error del 1% y α=0.01 en la tabla de distribución normal.

$$Z_{(0,01)} = -2.325$$

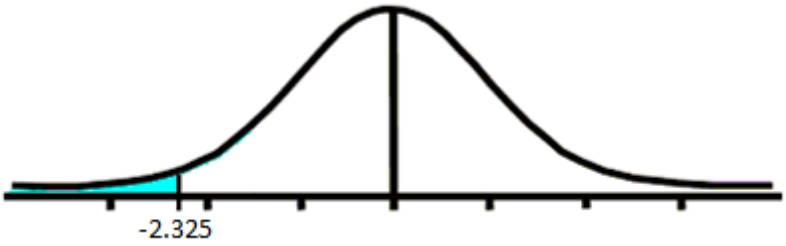


Figura 26: Z calculado con 99 % de confiabilidad.

Fuente: Elaboración propia

Reemplazando en la **ecuación 2** distribución normal para diferencia de muestras.

$$Z = \frac{(5786.46 - 5786.13) - 0}{\sqrt{\frac{26874.866^2}{100} + \frac{26874.834^2}{100}}}$$

$$\mathrm{Z} = \frac{0.33}{3800.68}$$

$$\mathrm{Z} = 0.00008683$$

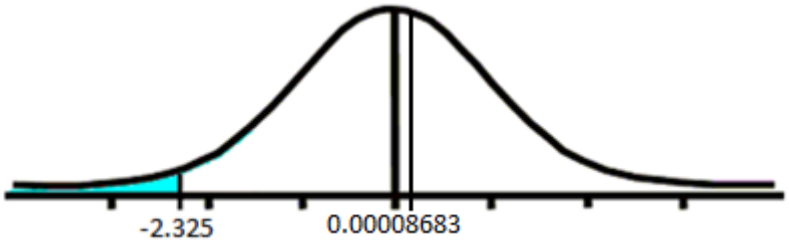


Figura 27: Z calculado vs Z de tabla

Fuente: Elaboración propia

Por lo que aceptamos la hipótesis nula $\mathrm{H_o}$ ya que él Z calculado está dentro del área de aceptación.

4.4.2. POSIBLES SALIDAS DE LAS FUNCIONES A MODIFICAR CONTRA LAS POSIBLES SALIDAS DE LA FUNCIONES A MODIFICADAS.

En el capítulo 4.1.3 se estudió las entradas y salidas de la función ShiftRows sin modificar y en el capítulo 4.2.3 se estudió las posibles entradas y salidas de la función ShiftRows modificada, de estas dos pruebas se obtuvieron los siguientes resultados.

- Para el algoritmo original Rijndael tenemos dos posibles estados finales para cualquier número estándar de rondas o tamaño de claves.
- Para la modificación del algoritmo Rijndael tenemos cuatro posibles estados finales después de la primera ronda, después de la segunda tendremos 16 posibles estados finales, podemos resumir que los posibles estados finales están dados en razón de 4^{Nr} donde Nr es el número de rondas del algoritmo.

Podemos afirmar que la modificación realizada nos ofrece un mayor rango de posibles salidas para cualquier tamaño de clave.

4.4.3. COMPLEJIDAD TEMPORAL DEL ALGORITMO ORIGINAL CONTRA LA MODIFICACIÓN DEL ALGORITMO RIJNDAEL.

Del análisis de la complejidad temporal de ambos algoritmos se obtuvo los siguientes resultados.

- El algoritmo Rijndael es un algoritmo de encriptación por bloques, realiza las mismas operaciones sin importar los caracteres de la clave y mucho menos de los caracteres del archivo a encriptar, bajo esta premisa, podemos asumir que la implementación de dicho algoritmo tiene una complejidad temporal $O(n)$, donde n es el tamaño del archivo

a encriptar; el análisis a detalle del análisis de la complejidad temporal de este algoritmo puede ser encontrada en el capítulo 4.1.

- La complejidad temporal del algoritmo Rijndael modificado esta descrita en el capítulo 4.3.1 y el capítulo 4.3.2 de donde encontramos que la complejidad temporal de la modificación del algoritmo Rijndael es $O(n)$.

Bajo la comparación de ambos análisis podemos asumir que la complejidad temporal de ambos algoritmos es similar, es quiere decir que para cualquier tamaño de archivo la proporción de tiempo de ejecución de ambos algoritmos variara de manera similar.

Análisis Asintótico del Algoritmo Original Rijndael	`Cipher(){`
	`  state = input;`
	`  KeyExpansion(k);` `O(KeyExpansion(k))`
	`  AddRoundKey(0);` `O(AddRoundKey(0))`
	
	`  for(round=1;round<Nr;round++){` `O(Nr(`
	`  SubBytes();` `O(SubBytes())`
	`  ShiftRows();` `O(ShiftRows())`
	`  MixColumns();` `O(MixColumns())`
	`  AddRoundKey(round);` `O(AddRoundKey(round))`
	`  }` `))`
	
	`  SubBytes();` `O(SubBytes())`
	`  ShiftRows();` `O(ShiftRows())`
	`  AddRoundKey(Nr);` `O(AddRoundKey(Nr))`
	
	`  Out = state;`
	
	`}`

Análisis asintótico de la Modificación del Algoritmo Rijndael	`Cipher(){`
	`  state = input;`
	`  KeyExpansion(k);` `O(KeyExpansion(k))`
	`  AddRoundKey(0);` `O(AddRoundKey(0))`
	
	`  for(round=1;round<Nr;round++){` `O(Nr(`
	`  SubBytes();` `O(SubBytes())`
	`  ShiftRows();` `O(ShiftRows())`
	`  MixColumns();` `O(MixColumns())`
	`  AddRoundKey(round);` `O(AddRoundKey(round))`
	`  }` `))`
	
	`  SubBytes();` `O(SubBytes())`
	`  ShiftRows();` `O(ShiftRows())`
	`  AddRoundKey(Nr);` `O(AddRoundKey(Nr))`
	
	`  Out = state;`
	`}`

Tabla 65: Análisis asintótico del algoritmo original vs la modificación

Elaboración: Propia

No se modificaron el orden de ejecución de las funciones por lo que el análisis asintótico de la función sigue siendo el mismo.

Análisis asintótico de la función KeyExpansion original

```
KeyExpansion () {                                                    
  int i,j;                                                           
  unsigned char temp[4],k;                                           
                                                                     
  for (i=0; i<Nk; i++){                                              O(Nk(
    RoundKey[i*4]=Key[i*4];                                            O(1)
    RoundKey[i*4+1]=Key[i*4+1];                                        O(1)
    RoundKey[i*4+2]=Key[i*4+2];                                        O(1)
    RoundKey[i*4+3]=Key[i*4+3];}                                       O(1)))
                                                                     
  while (i < (Nb * (Nr+1))){                                         O(Nb * (Nr + 1)(
    for(j=0;j<4;j++){                                                  O(4)(
      temp[j]=RoundKey[(i-1) * 4 + j];                                   O(1)
    }                                                                  )
    if (i % Nk == 0){                                                  O(1)
      // Función RotWord ();                                           
      {  k = temp[0];                                                  O(1)
        temp[0] = temp[1];                                             O(1)
        temp[1] = temp[2];                                             O(1)
        temp[2] = temp[3];                                             O(1)
        temp[3] = k;}                                                  O(1)
      // Función SubWord ();                                           
      {  temp[0]=getSBoxValue(temp[0]);                                O(getSBoxValue)
        temp[1]=getSBoxValue(temp[1]);                                 O(getSBoxValue)
        temp[2]=getSBoxValue(temp[2]);                                 O(getSBoxValue)
        temp[3]=getSBoxValue(temp[3]);}                                O(getSBoxValue)
      temp[0] =  temp[0] ^ Rcon[i/Nk];                                 O(1)
    }                                                                  
    else if (Nk > 6 && i % Nk == 4){                                   O(2)
      // Función Subword()                                             
      {  temp[0]=getSBoxValue(temp[0]);                                O(getSBoxValue)
        temp[1]=getSBoxValue(temp[1]);                                 O(getSBoxValue)
        temp[2]=getSBoxValue(temp[2]);                                 O(getSBoxValue)
        temp[3]=getSBoxValue(temp[3]);}                                O(getSBoxValue)
    }                                                                  
    RoundKey[i*4+0] = RoundKey[(i-Nk)*4+0] ^ temp[0];                  O(1)
    RoundKey[i*4+1] = RoundKey[(i-Nk)*4+1] ^ temp[1];                  O(1)
    RoundKey[i*4+2] = RoundKey[(i-Nk)*4+2] ^ temp[2];                  O(1)
    RoundKey[i*4+3] = RoundKey[(i-Nk)*4+3] ^ temp[3];                  O(1)
    i++;                                                               
  }                                                                    
}                                                                      ))
```

Continua →

Análisis asintótico de la función KeyExpansion modificada

```
KeyExpansion () {                                                    
  int i,j;                                                           
  unsigned char temp[4],k;                                           
                                                                     
  for (i=0; i<Nk; i++){                                              O(Nk)(
    RoundKey[i*4]=Key[i*4];                                           O(1)
    RoundKey[i*4+1]=Key[i*4+1];                                       O(1)
    RoundKey[i*4+2]=Key[i*4+2];                                       O(1)
    RoundKey[i*4+3]=Key[i*4+3];                                       O(1)
  }                                                                  )
                                                                     
  while (i < (Nb * (Nr+1))){                                         O(Nb * (Nr + 1))(
    for(j=0;j<4;j++){                                                 O(4)(
      temp[j]=RoundKey[(i-1) * 4 + j];                                 O(1)
    }                                                                 )
                                                                     
    if (i % Nk == 0){                                                O(1) +
      {// Función RotWord ();
        k = temp[0];
        temp[0] = temp[1];
        temp[1] = temp[2];
        temp[2] = temp[3];
        temp[3] = k;
      }
      {// Función SubWord ();
        temp[0]=getSBoxValue(temp[0]);
        temp[1]=getSBoxValue(temp[1]);
        temp[2]=getSBoxValue(temp[2]);
        temp[3]=getSBoxValue(temp[3]);
      }
      temp[0] =  temp[0] ^ Rcon[i/Nk];
    }
    else if (Nk > 6 && i % Nk == 4){                                 O(1) +
      {// Función Subword()
        temp[0]=getSBoxValue(temp[0]);
        temp[1]=getSBoxValue(temp[1]);
        temp[2]=getSBoxValue(temp[2]);
        temp[3]=getSBoxValue(temp[3]);
      }
    }
    RoundKey[i*4+0] = RoundKey[(i-Nk)*4+0] ^ temp[0];                O(1)
    RoundKey[i*4+1] = RoundKey[(i-Nk)*4+1] ^ temp[1];                O(1)
    RoundKey[i*4+2] = RoundKey[(i-Nk)*4+2] ^ temp[2];                O(1)
    RoundKey[i*4+3] = RoundKey[(i-Nk)*4+3] ^ temp[3];                O(1)
    ShiftMoves[i-Nk]   =   (RoundKey[i*4+0]   +   RoundKey[i*4+1]   +
RoundKey[i*4+2] + RoundKey[i*4+3]) % 2;                               O(1)
  i++;                                                               O(1)
  }                                                                  )
}
```

Tabla 66: Análisis asintótico de la función original KeyExpansion vs la modificación de esta

Elaboración: Propia

Por lo que las ecuaciones finales estarían dadas por:

Análisis asintótico de la función KeyExpansion original	**Análisis asintótico de la función KeyExpansion modificada**
$O(Nk) + O\left(Nb\,(Nr+1)\left(O(4)*O(1) + O\left(\frac{5\ +\ 4(\text{getSBoxValue})+1}{Nk}\right) + O\left(\frac{4(\text{getSBoxValue})}{Nk}\right)+O(4)\right)\right)$	$O(Nk) + O\left(Nb(Nr+1)\left(O(4) + O\left(\frac{5\ +\ 4(\text{getSBoxValue})}{Nk}\right) + O\left(\frac{\text{getSBoxValue}}{4}\right)+O(5)\right)\right)$

Tabla 67: Ecuación del análisis asintótico de la función KeyExpansion vs la modificación

Elaboración: Propia

Por conocer los valores de Nr, Nb y Nk podemos reducir las ecuaciones a:

Análisis asintótico de la función KeyExpansion original	**Análisis asintótico de la función KeyExpansion modificada**
$O(k)$	$O(k)$

Tabla 68: Ecuaciones asintóticas de las funciones KeyExpansion reducidas.

Elaboración: Propia

Análisis asintótico de la función ShiftRows original

```
ShiftRows(){
  unsigned char temp;

  // Rotar la segunda fila 1 pos a la izquierda
  temp=state[1][0];                                   O(1)
  state[1][0]=state[1][1];                            O(1)
  state[1][1]=state[1][2];                            O(1)
  state[1][2]=state[1][3];                            O(1)
  state[1][3]=temp;                                   O(1)

  // Rotar la tercera fila 2 pos a la izquierda
  temp=state[2][0];                                   O(1)
  state[2][0]=state[2][2];                            O(1)
  state[2][2]=temp;                                   O(1)
  temp=state[2][1];                                   O(1)
  state[2][1]=state[2][3];                            O(1)
  state[2][3]=temp;                                   O(1)

  // Rotar la cuarta fila 1 pos a la izquierda
  temp=state[3][0];                                   O(1)
  state[3][0]=state[3][3];                            O(1)
  state[3][3]=state[3][2];                            O(1)
  state[3][2]=state[3][1];                            O(1)
  state[3][1]=temp;                                   O(1)
}
```

Análisis asintótico de la función ShiftRows modificada

```
ShiftRows(int round){
 char temp;

 if(ShiftMoves[(4*round)-3]==0){                      O(1) + (
  temp=state[1][0];                                    O(1)
  state[1][0]=state[1][1];                             O(1)
  state[1][1]=state[1][2];                             O(1)
  state[1][2]=state[1][3];                             O(1)
  state[1][3]=temp;                                    O(1)
 }                                                    )
 else if(ShiftMoves[(4*round)-3]==1){                 O(1) + (
  temp=state[1][3];                                    O(1)
  state[1][3]=state[1][2];                             O(1)
  state[1][2]=state[1][1];                             O(1)
  state[1][1]=state[1][0];                             O(1)
  state[1][0]=temp;                                    O(1)
 }                                                    )
 // Rotate second row 2 columns to left or right
 temp=state[2][0];                                    O(1)
 state[2][0]=state[2][2];                             O(1)
 state[2][2]=temp;                                    O(1)
 temp=state[2][1];                                    O(1)
 state[2][1]=state[2][3];                             O(1)
 state[2][3]=temp;                                    O(1)

 if(ShiftMoves[(4*round)-2]==0){                      O(1) + (
  temp=state[3][0];                                    O(1)
  state[3][0]=state[3][3];                             O(1)
  state[3][3]=state[3][2];                             O(1)
  state[3][2]=state[3][1];                             O(1)
  state[3][1]=temp;                                    O(1)
 }                                                    )
 else if(ShiftMoves[(4*round)-2]==1){                 O(1) + (
  temp=state[3][0];                                    O(1)
  state[3][0]=state[3][1];                             O(1)
  state[3][1]=state[3][2];                             O(1)
  state[3][2]=state[3][3];                             O(1)
  state[3][3]=temp;                                    O(1)
 }                                                    )
}
```

Tabla 69: Análisis asintótico de la función original ShifRows vs la modificación de esta

Fuente: Elaboración propia

Por lo que las ecuaciones de ambos algoritmos quedan reducidas a:

Análisis asintótico de la función KeyExpansion original	**Análisis asintótico de la función KeyExpansion modificada**
$O(5) + O(6) + O(5)$	$O(1) + O(5) + O(1) + O(5) + O(6) + O(1) + O(5) + O(1) + O(5)$

Tabla 70: Ecuación del análisis asintótico de la función ShiftRows vs la modificación

Elaboración: Propia

Podemos asumir que estas ecuaciones se pueden reducir a:

Análisis asintótico de la función KeyExpansion original	**Análisis asintótico de la función KeyExpansion modificada**
$O(k)$	$O(k)$

Tabla 71: Ecuaciones asintóticas de las funciones ShiftRows reducidas.

Elaboración: Propia

4.5. INTERPRETACIÓN DE RESULTADOS

En esta sección se interpretara y explicara de manera clara los resultados encontrados en los subcapítulos anteriores.

4.5.1. DEL ANÁLISIS DEL ALGORITMO ORIGINAL RIJNDAEL.

En el análisis de la complejidad temporal del algoritmo original Rijndael (capítulo 4.1.1) se encontró que la complejidad temporal de este es $O(n)$, ya que la encriptación a realizar es una encriptación por bloques, el algoritmo toma un bloque de 128 bits lo encripta y continua con el siguiente bloque de 128 bits hasta completar la totalidad el tamaño del archivo, las operaciones que realiza el

algoritmo dependen directamente del tamaño de la clave, ya que conocemos todos los tamaños de clave a ser utilizados en el algoritmo podemos asumir que el impacto del tamaño de la clave en el tiempo de ejecución como $O(k)$; esto quiere decir que el tiempo que demora en ejecutarse la encriptación depende directamente del tamaño del archivo a encriptar.

Del análisis de la entropía del algoritmo original Rijndael (capítulo 4.1.2) podemos afirmar que la independencia matemática entre los caracteres del algoritmo es muy alta ya que los diferentes algoritmos de compresión aplicados a los archivos previamente encriptados no pudieron encontrar razón matemática en ellos, tampoco crear un árbol de caracteres para su compresión, para el peor caso; para el mejor de los casos, cuando los caracteres de la clave es repetitiva y los archivos a encriptar también son repetitivos se encontró una reducción del tamaño al comprimir los archivos, ya que el algoritmo Rijndael es un algoritmo de encriptación por bloques y para el segundo bloque a encriptar este repite el mismo proceso que se realizó con el primer bloque, hasta finalizar con todo el archivo.

Del análisis del número de posibles salidas de la función ShiftRows (capítulo 4.1.3) podemos encontrar que la dispersión de datos ofrecida por esta función es muy pequeña.

4.5.2. DEL DISEÑO DE LAS MODIFICACIONES DEL ALGORITMO ORIGINAL

La modificación de las funciones KeyExpansion y ShiftRows nos ofrece una mayor dispersión de los bits de estado ya que estas funciones ahora dependen de las subclaves generadas por la función KeyExpansion y estas subclaves tienen influencia directa en los movimientos de la función ShiftRows.

4.5.3. DE LA IMPLEMENTACIÓN Y ANALISIS DE LA MODIFICACIÓN DEL ALGORITMO RIJNDAEL.

La implementación de la modificación del algoritmo Rijndael se realizó en el lenguaje c++ siguiendo la documentación oficial publicada por el FIPS publicada en el 2001.

En el análisis de la complejidad temporal de la modificación de la función KeyExpansion (capítulo 4.3.1) se encontró que esta es igual a $O(k)$ ya que esta función conserva sus propiedades originales, adicional a dichas funciones, opera y adiciona estos valores al array ShiftMoves.

En el análisis de la complejidad temporal de la modificación de la función ShiftRows (capítulo 4.3.2) se encontró que esta es igual a $O(k)$, la nueva función verifica los valores del array ShiftMoves de acuerdo a estos parámetros mueve las filas de la matriz de estado, con esta modificación podemos asegurar que la función ShiftRows ofrece una mejor distribución de los datos.

En el análisis de la entropía de las salidas del algoritmo Rijndael modificado, para realizar esta prueba se utilizó los mismos archivos, y claves utilizados en el análisis de entropía del algoritmo original (capitulo 4.1.2); se encontró resultados similares que con el algoritmo original.

Para el peor caso al comprimir los archivos encriptados con los algoritmos de compresión deflate y LZMA, se obtuvo un incremento en el tamaño de los archivos igual que al comprimir archivos encriptados con el algoritmo original.

Para el mejor caso, al comprimir archivos con el algoritmo LZMA se obtuvo una pequeña diferencia en el tamaño de los archivos con respecto a los tamaños obtenidos al comprimir archivos encriptados con el algoritmo original, pero

utilizando el algoritmo deflate se obtuvo una diferencia de tamaño más significativa con respecto a los archivos comprimidos con el algoritmo original.

4.5.4. DEL ANÁLISIS Y COMPARACIÓN DE LAS NUEVAS FUNCIONES CON LAS FUNCIONES ORIGINALES.

En la comparación de la entropía del algoritmo original Rijndael con la modificación de este (capitulo 4.4.1) se encontró que existe una diferencia en los tamaños de archivos entre los archivos generados por el algoritmo original Rijndael y los generados por la modificación de este, para el peor caso no se encontró diferencias en el tamaño del archivo ya que el algoritmo original y la modificación de este ofrecen una alta independencia matemática entre los caracteres de los archivos una vez encriptados, para el mejor caso una vez comprimidos los archivos se encontró una reducción de tamaño en ambos algoritmos ya que estos son algoritmos de encriptación por bloques, toman bloques de 128 bits, los procesan y continúan con otro bloque de 128 bits hasta completar el tamaño total del archivo utilizando las mismas subclaves generadas en la primera ronda, de esta manera utilizando claves repetitivas y archivos repetitivos los primeros 128 bits se repetirán hasta cubrir el total del archivo, por esta razón los archivos encriptados con ambos algoritmos al ser comprimidos reducen su tamaño, al ser claves y archivos repetitivos, podemos analizar de manera más clara el impacto del algoritmo en estos archivos.

Realizando la comparación de tamaño de los archivos comprimidos para el mejor caso con el algoritmo LZMA encontramos una pequeña diferencia en el tamaño entre el algoritmo original Rijndael y la modificación propuesta a favor del algoritmo Rijndael modificado, esta diferencia se incrementa al utilizar el

algoritmo deflate. Para comprobar si esta diferencia es una diferencia al azar o una certeza probabilística se utilizó la distribución normal para diferencia de medias ya que el tamaño de las muestras es mayor muy grande para ser analizado con la distribución t de student, con un nivel de confianza del 99% y margen de error del 1%, se probó que la diferencia de tamaño entre las muestras puede ser repetida, de esta prueba podemos deducir q la modificación ofrece una mayor entropía en los archivos encriptados.

Del análisis de la cantidad de posible salidas de las funciones a modificar del algoritmo Rijndael original contra la modificación de este (capítulo 4.4.2) podemos concluir que al modificar las funciones KeyExpansion y ShiftRows obtendremos una mayor entropía en las salidas de la función ShiftRows, esta modificaron se ve reflejada en los archivos encriptados y comprimidos.

Al analizar la complejidad temporal del algoritmo original Rijndael y de la modificación de este se encontró que esta es $O(n)$, la complejidad temporal de ambos algoritmos es similar, esto quiere decir que para cualquier tamaño de archivo la proporción de tiempo de ejecución de ambos algoritmos variara de manera similar.

CONCLUSIONES

PRIMERO: La modificación propuesta no incrementa en la complejidad temporal general del algoritmo, por lo que es posible poder implementar el cambio al algoritmo sin que el tiempo de ejecución incremente.

SEGUNDO: La función original Rijndael ofrece una gran entropía entre los archivos encriptados, y la complejidad temporal depende directamente del tamaño del archivo.

TERCERO: La modificación e implementación de la función ShiftRow () nos asegura una mejor distribución de los datos ya que con esta modificación la difusión de caracteres es dependiente de las sub-claves generadas a partir de la clave.

CUARTO: Con la modificación de la función ShiftRow () no existe ningún tipo de perdida de datos al encriptar y desencriptar datos, ya que se modificó la forma en que se distribuyen las posiciones de los datos y no se realiza ninguna operación directa con estos.

QUINTO: La implementación de la modificación de la función ShiftRow() en el algoritmo Rijndael asegura una mayor independencia matemática en la generación de textos cifrados.

SUGERENCIAS

PRIMERO: A próximos investigadores, realizar un estudio más profundo sobre las medidas de entropía de los datos, ya que es difícil probar que un algoritmo o función genera datos más complejos de descifrar que otro.

SEGUNDO: Ampliar el tamaño de las claves, para la encriptación incrementa la seguridad del sistema de cifrado, ya que las pruebas por fuerza bruta se basan en cuantas claves por segundo pueden ser ejecutadas en un algoritmo, ya que mientras más grande sea el tamaño de la clave más combinaciones serán probadas.

TERCERO: Es posible implementar un algoritmo de encriptación por bloques de manera que para cada bloque las subclaves a utilizar sean diferentes a las usada en el bloque anterior, de esta forma se asegurara la entropía cuando la clave y el archivo a encriptar son de caracteres repetitivos.

BIBLIOGRAFÍA

C. LIBERATORI, M. (2006). *Desarrollo de encriptado en AES en FPGA.* Ciudad de la Plata – Argentina.

ANGEL ANGEL, J. D. (2005). *AES - Advanced Encryption Standard VERSIÓN 2005.* México.

BASURTO BECERRA, A. J. (2015). *Aspectos de seguridad de bitcoin y su aplicación en una alternativa de infraestructura de llave pública.* MÉXICO.

Borja, L., & De Los Santos, S. (24 de Agosto de 2011). *Hispasec.* Obtenido de http://unaaldia.hispasec.com/2011/08/el-cifrado-aes-esta-roto-o-no.html

Bruno Castañeda, C., & Caballero Gil, P. (1997). Sistemas criptograficos de sustitución. *Números*, 18.

CCM.net - Kioskea ES. (diciembre de 2015). Recuperado el 4 de enero de 2016, de http://es.ccm.net/contents/130-introduccion-al-cifrado-mediante-des

Coronado, S. P. (Febrero de 2001). *Artículos con Clase*. Obtenido de Artículos con Clase web site: http://articulos.conclase.net/?tema=algoritmos&art=huffman&pag=000#inicio

FIGUEROA GARCÍA, R. (2007). *Matematica basica 2 vectores y matrices con numeros complejos .* Lima - Peru: Ediciones RFG.

GARCÍA MENDEZ, P. S. (2011). *Descripción polinomial de los sistemas de cifrado DES y AES.* México, D.F.

Gonzalez, M. L. (2008). *Página Académica Manuel Lobos Gonzáles*. Obtenido de Página Académica Manuel Lobos Gonzáles: http://www.mey.cl/clases/clases_2miv2.ppt

LEON LOMPARTE, K. R. (2005). *Encriptacion RSA.* Lima – Perú.

Loayza, M. A. (2012). *investigación cualitativa y cuantitativa en educación.* Puno: Corporación Merú E.I.R.L.

Menendez, F. J. (2009). *Georreferenciación de Cartografia: Datos Raster y Vectoriales.* Madrid españa: EOSGIS S.L.

Newbold, P., Carlson, W., & Thorne, B. (2008). *Estadística para Administración y Economía.* Madrid: PEARSON EDUCACIÓN, S.A.

Paar, C., & Pelzl, J. (2010). *Understanding Cryptography.* Berlin: Springer-Verlag.

Rawal, S. (2016). Advanced Encryption Standard (AES) and It's Working. *International Research Journal of Engineering and Technology (IRJET)*, 1165.

ROA BUENDÍA, J. F. (2013). *Seguridad informática.* Madrid - España: McGraw-Hill/Interamericana de España, S.L.

TALENS-OLIAG, S. (2003). *Introducción a la Criptología.* Valencia - España.

VALENZUELA RUZ, V. (2003). *Manual análisis de algoritmos.* Copiapó - Chile.

ANEXOS

Anexo 1: Implementación del algoritmo de encriptación Rijndael original en c++ comentado

```
// Incluimos stdio.h para entradas/salidas estandar.
#include <stdio.h>

// Número de columnas de la matriz de estado, es constante
#define Nb 4

// Numero de rondas inicialmente 0, su valor cambia en ejecucion
int Nr=0;

// Número de palabras de 32 bits de la clave, su valor cambia en
// ejecucion.

int Nk=0;

// in - es el array que contiene el texto plano a ser encriptado.
// out - es el array que contiene la clave para encriptar.
// state - es el array que contiene los resultados intermedios durante
// la encriptación.

unsigned char in[16], out[16], state[4][4];

// el array que contiene las subclaves para cada ronda.
unsigned char RoundKey[240];

// la clave de entrada
unsigned char Key[32];

int getSBoxValue(int num){
 int sbox[256] =   {
 //0    1    2    3    4    5    6    7    8    9    A    B    C    D    E    F
 0x63,0x7c,0x77,0x7b,0xf2,0x6b,0x6f,0xc5,0x30,0x01,0x67,0x2b,0xfe,0xd7,0xab,0x76, //0
 0xca,0x82,0xc9,0x7d,0xfa,0x59,0x47,0xf0,0xad,0xd4,0xa2,0xaf,0x9c,0xa4,0x72,0xc0, //1
 0xb7,0xfd,0x93,0x26,0x36,0x3f,0xf7,0xcc,0x34,0xa5,0xe5,0xf1,0x71,0xd8,0x31,0x15, //2
 0x04,0xc7,0x23,0xc3,0x18,0x96,0x05,0x9a,0x07,0x12,0x80,0xe2,0xeb,0x27,0xb2,0x75, //3
 0x09,0x83,0x2c,0x1a,0x1b,0x6e,0x5a,0xa0,0x52,0x3b,0xd6,0xb3,0x29,0xe3,0x2f,0x84, //4
 0x53,0xd1,0x00,0xed,0x20,0xfc,0xb1,0x5b,0x6a,0xcb,0xbe,0x39,0x4a,0x4c,0x58,0xcf, //5
 0xd0,0xef,0xaa,0xfb,0x43,0x4d,0x33,0x85,0x45,0xf9,0x02,0x7f,0x50,0x3c,0x9f,0xa8, //6
 0x51,0xa3,0x40,0x8f,0x92,0x9d,0x38,0xf5,0xbc,0xb6,0xda,0x21,0x10,0xff,0xf3,0xd2, //7
 0xcd,0x0c,0x13,0xec,0x5f,0x97,0x44,0x17,0xc4,0xa7,0x7e,0x3d,0x64,0x5d,0x19,0x73, //8
 0x60,0x81,0x4f,0xdc,0x22,0x2a,0x90,0x88,0x46,0xee,0xb8,0x14,0xde,0x5e,0x0b,0xdb, //9
 0xe0,0x32,0x3a,0x0a,0x49,0x06,0x24,0x5c,0xc2,0xd3,0xac,0x62,0x91,0x95,0xe4,0x79, //A
 0xe7,0xc8,0x37,0x6d,0x8d,0xd5,0x4e,0xa9,0x6c,0x56,0xf4,0xea,0x65,0x7a,0xae,0x08, //B
 0xba,0x78,0x25,0x2e,0x1c,0xa6,0xb4,0xc6,0xe8,0xdd,0x74,0x1f,0x4b,0xbd,0x8b,0x8a, //C
 0x70,0x3e,0xb5,0x66,0x48,0x03,0xf6,0x0e,0x61,0x35,0x57,0xb9,0x86,0xc1,0x1d,0x9e, //D
 0xe1,0xf8,0x98,0x11,0x69,0xd9,0x8e,0x94,0x9b,0x1e,0x87,0xe9,0xce,0x55,0x28,0xdf, //E
 0x8c,0xa1,0x89,0x0d,0xbf,0xe6,0x42,0x68,0x41,0x99,0x2d,0x0f,0xb0,0x54,0xbb,0x16};//F
 return sbox[num];
}

//el conjunto de valores constantes Rcon[i], contiene los valores dados
//por x a la potencia (i-1) comenzando con la potencia de x (x es denotado
//como {02}) en el campo GF(2^8)
//nótese que se comienza de 1 y no 0).
int Rcon[255] = {
 0x8d,0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80,0x1b,0x36,0x6c,0xd8,0xab,0x4d,0x9a,
```

```
 0x2f,0x5e,0xbc,0x63,0xc6,0x97,0x35,0x6a,0xd4,0xb3,0x7d,0xfa,0xef,0xc5,0x91,0x39,
 0x72,0xe4,0xd3,0xbd,0x61,0xc2,0x9f,0x25,0x4a,0x94,0x33,0x66,0xcc,0x83,0x1d,0x3a,
 0x74,0xe8,0xcb,0x8d,0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80,0x1b,0x36,0x6c,0xd8,
 0xab,0x4d,0x9a,0x2f,0x5e,0xbc,0x63,0xc6,0x97,0x35,0x6a,0xd4,0xb3,0x7d,0xfa,0xef,
 0xc5,0x91,0x39,0x72,0xe4,0xd3,0xbd,0x61,0xc2,0x9f,0x25,0x4a,0x94,0x33,0x66,0xcc,
 0x83,0x1d,0x3a,0x74,0xe8,0xcb,0x8d,0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80,0x1b,
 0x36,0x6c,0xd8,0xab,0x4d,0x9a,0x2f,0x5e,0xbc,0x63,0xc6,0x97,0x35,0x6a,0xd4,0xb3,
 0x7d,0xfa,0xef,0xc5,0x91,0x39,0x72,0xe4,0xd3,0xbd,0x61,0xc2,0x9f,0x25,0x4a,0x94,
 0x33,0x66,0xcc,0x83,0x1d,0x3a,0x74,0xe8,0xcb,0x8d,0x01,0x02,0x04,0x08,0x10,0x20,
 0x40,0x80,0x1b,0x36,0x6c,0xd8,0xab,0x4d,0x9a,0x2f,0x5e,0xbc,0x63,0xc6,0x97,0x35,
 0x6a,0xd4,0xb3,0x7d,0xfa,0xef,0xc5,0x91,0x39,0x72,0xe4,0xd3,0xbd,0x61,0xc2,0x9f,
 0x25,0x4a,0x94,0x33,0x66,0xcc,0x83,0x1d,0x3a,0x74,0xe8,0xcb,0x8d,0x01,0x02,0x04,
 0x08,0x10,0x20,0x40,0x80,0x1b,0x36,0x6c,0xd8,0xab,0x4d,0x9a,0x2f,0x5e,0xbc,0x63,
 0xc6,0x97,0x35,0x6a,0xd4,0xb3,0x7d,0xfa,0xef,0xc5,0x91,0x39,0x72,0xe4,0xd3,0xbd,
 0x61,0xc2,0x9f,0x25,0x4a,0x94,0x33,0x66,0xcc,0x83,0x1d,0x3a,0x74,0xe8,0xcb};

// Esta función produce Nb(Nr+1) llaves de ronda. Las llaves de ronda
son usadas en cada ronda para encriptar el estado.
void KeyExpansion(){
 int i,j;
 unsigned char temp[4],k;

 // la primera clave de ronda es ella misma.
 for(i=0;i<Nk;i++){
  RoundKey[i*4]=Key[i*4];
  RoundKey[i*4+1]=Key[i*4+1];
  RoundKey[i*4+2]=Key[i*4+2];
  RoundKey[i*4+3]=Key[i*4+3];
 }
 //todas las demás claves de ronda son encontradas de la clave de ronda
 //anterior.
 while (i < (Nb * (Nr+1))){
  for(j=0;j<4;j++){
   temp[j]=RoundKey[(i-1) * 4 + j];
  }
  if (i % Nk == 0){
 // esta función rota 4 bytes de una palabra a la izquierda.
 // convierte [a0,a1,a2,a3] a [a1,a2,a3,a0]
 // Function RotWord()
   {
    k = temp[0];
    temp[0] = temp[1];
    temp[1] = temp[2];
    temp[2] = temp[3];
    temp[3] = k;
   }
 // SubWord() Toma una palabra de entrada de cuatro bytes y aplica el
 // S-box a cada uno de los cuatro bytes.
 // Function Subword()
   {
    temp[0]=getSBoxValue(temp[0]);
    temp[1]=getSBoxValue(temp[1]);
    temp[2]=getSBoxValue(temp[2]);
    temp[3]=getSBoxValue(temp[3]);
   }
   temp[0] =  temp[0] ^ Rcon[i/Nk];
  }

  else if (Nk > 6 && i % Nk == 4){
  // Function Subword()
   {
    temp[0]=getSBoxValue(temp[0]);
    temp[1]=getSBoxValue(temp[1]);
```

```
      temp[2]=getSBoxValue(temp[2]);
      temp[3]=getSBoxValue(temp[3]);
     }
    }
    RoundKey[i*4+0] = RoundKey[(i-Nk)*4+0] ^ temp[0];
    RoundKey[i*4+1] = RoundKey[(i-Nk)*4+1] ^ temp[1];
    RoundKey[i*4+2] = RoundKey[(i-Nk)*4+2] ^ temp[2];
    RoundKey[i*4+3] = RoundKey[(i-Nk)*4+3] ^ temp[3];
    i++;
  }
}

// Esta función agrega cada llave de ronda al estado.
// la llave de donda es agregada al estado con una función XOR.
void AddRoundKey(int round){
  int i,j;
  for(i=0;i<4;i++){
    for(j=0;j<4;j++){
      state[j][i] ^= RoundKey[round * Nb * 4 + i * Nb + j];
    }
  }
}

// La función SubBytes sustituye los valores de la matris de estado con
// valores de una tabla S-Box.
void SubBytes(){
  int i,j;
  for(i=0;i<4;i++){
    for(j=0;j<4;j++){
      state[i][j] = getSBoxValue(state[i][j]);
    }
  }
}

// La función ShiftRows mueve las filas de estado a la izquierda.
// para cada fila existe un diferente desplazamiento
// desplazamiento = munero de fila.
void ShiftRows(){
  unsigned char temp;
  // La primera fila es movida 1 columna a la derecha
  temp=state[1][0];
  state[1][0]=state[1][1];
  state[1][1]=state[1][2];
  state[1][2]=state[1][3];
  state[1][3]=temp;

  // La segunda fila es movida 2 columnas a la derecha
  temp=state[2][0];
  state[2][0]=state[2][2];
  state[2][2]=temp;
  temp=state[2][1];
  state[2][1]=state[2][3];
  state[2][3]=temp;

  // La tercera fila es movida 3 columnas a la derecha
  temp=state[3][0];
  state[3][0]=state[3][3];
  state[3][3]=state[3][2];
  state[3][2]=state[3][1];
  state[3][1]=temp;
}
```

```
// xtime es un macro que encuentra el producto de {02} y el argumento
// xtime modulo {1b}
#define xtime(x)   ((x<<1) ^ (((x>>7) & 1) * 0x1b))
// La función MixColumns mezcla las columnas de la matriz de estado
void MixColumns(){
 int i;
 unsigned char Tmp,Tm,t;
 for(i=0;i<4;i++){
  t=state[0][i];
  Tmp = state[0][i] ^ state[1][i] ^ state[2][i] ^ state[3][i];
  Tm = state[0][i] ^ state[1][i];
  Tm = xtime(Tm); state[0][i] ^= Tm ^ Tmp;
  Tm = state[1][i] ^ state[2][i];
  Tm = xtime(Tm); state[1][i] ^= Tm ^ Tmp;
  Tm = state[2][i] ^ state[3][i];
  Tm = xtime(Tm); state[2][i] ^= Tm ^ Tmp;
  Tm = state[3][i] ^ t;
  Tm = xtime(Tm);
  state[3][i] ^= Tm ^ Tmp;
 }
}

// Cipher es la función principal que encripta el texto plano.
void Cipher(){
 int i,j,round=0;
 //copiar el texto plano al array de estado.
 for(i=0;i<4;i++){
  for(j=0;j<4;j++){
   state[j][i] = in[i*4 + j];
  }
 }

 // Agregar la primera clave de rondas al estado antes de iniciar las
 //  rondas.
 AddRoundKey(0);

 // Habrá Nr rondas.
 // Las primeras Nr-1 rondas son identicas.
 // Estas Nr-1 rondas se ejecutan en el bucle a continuacion.
 for(round=1;round<Nr;round++){
  SubBytes();
  ShiftRows();
  MixColumns();
  AddRoundKey(round);
 }

 // La última ronda se da a continuacion.
 // la función MixColumns no se ejecuta en la última ronda.
 SubBytes();
 ShiftRows();
 AddRoundKey(Nr);

 // El proceso de encriptación ha terminado.
 // Copiamos la matriz de estado a un array de salida.
 for(i=0;i<4;i++){
  for(j=0;j<4;j++){
   out[i*4+j]=state[j][i];
  }
```

```
 }
}

int main(int argc, char * argv[]){
 int i;

 // recibimos el tamaño de la clave.
 while(Nr!=128 && Nr!=192 && Nr!=256){
//  Para propósitos demostrativos el tamaño de la clave es de 128 bits.
//  Para ingresar un tamaño de clave descomentar la función scanf.
//  scanf("%d",&Nr);
   Nr=128;
 }

 // Calculamos Nk y Nr del valor recibido.
 Nk = Nr / 32;
 Nr = Nk + 6;
//leemos la clave y el archivo a ser encriptado.
 memcpy(Key, argv[1], sizeof Key);
//memcpy(in, argv[2], 16);
    FILE *input=fopen(argv[2],"r");
//el archivo encriptado saldrá con el nombre de output.txt
    outputm=fopen("output.txt","wb");

    KeyExpansion();
    int q=0;
    while(fread(in,sizeof(char),16,input)== 16){
        Cipher();
    }
 return 0;
}
```

Anexo 2: Implementación de la modificación algoritmo de encriptación Rijndael en c++ comentado

```
// Incluimos las librerías necesarias.
#include <stdio.h>
#include <iostream>
#include <cstring>
#include <stdlib.h>

// Número de columnas de la matriz de estado, es constante
#define Nb 4

// Numero de rondas inicialmente 0, su valor cambia en ejecucion
int Nr=0;

// Número de palabras de 32 bits de la clave, su valor cambia en
// ejecucion.
int Nk=0;

//variable del archivo de salida
FILE *outputm;

// in - es el array que contiene el texto plano a ser encriptado.
// out - es el array que contiene la clave para encriptar.
// state - es el array que contiene los resultados intermedios durante
// la encriptación.
```

```
unsigned char in[16], out[16], state[4][4];


// El array que contiene las subclaves para cada ronda.
unsigned char RoundKey[240];

// ShiftMoves almacenara las movidas a ser usadas en la función ShiftRows
unsigned char ShiftMoves[240];

// La clave de entrada
unsigned char Key[32];

int getSBoxValue(int num){
 int sbox[256] =   {
 //0    1    2    3    4    5    6    7    8    9    A    B    C    D    E    F
 0x63,0x7c,0x77,0x7b,0xf2,0x6b,0x6f,0xc5,0x30,0x01,0x67,0x2b,0xfe,0xd7,0xab,0x76, //0
 0xca,0x82,0xc9,0x7d,0xfa,0x59,0x47,0xf0,0xad,0xd4,0xa2,0xaf,0x9c,0xa4,0x72,0xc0, //1
 0xb7,0xfd,0x93,0x26,0x36,0x3f,0xf7,0xcc,0x34,0xa5,0xe5,0xf1,0x71,0xd8,0x31,0x15, //2
 0x04,0xc7,0x23,0xc3,0x18,0x96,0x05,0x9a,0x07,0x12,0x80,0xe2,0xeb,0x27,0xb2,0x75, //3
 0x09,0x83,0x2c,0x1a,0x1b,0x6e,0x5a,0xa0,0x52,0x3b,0xd6,0xb3,0x29,0xe3,0x2f,0x84, //4
 0x53,0xd1,0x00,0xed,0x20,0xfc,0xb1,0x5b,0x6a,0xcb,0xbe,0x39,0x4a,0x4c,0x58,0xcf, //5
 0xd0,0xef,0xaa,0xfb,0x43,0x4d,0x33,0x85,0x45,0xf9,0x02,0x7f,0x50,0x3c,0x9f,0xa8, //6
 0x51,0xa3,0x40,0x8f,0x92,0x9d,0x38,0xf5,0xbc,0xb6,0xda,0x21,0x10,0xff,0xf3,0xd2, //7
 0xcd,0x0c,0x13,0xec,0x5f,0x97,0x44,0x17,0xc4,0xa7,0x7e,0x3d,0x64,0x5d,0x19,0x73, //8
 0x60,0x81,0x4f,0xdc,0x22,0x2a,0x90,0x88,0x46,0xee,0xb8,0x14,0xde,0x5e,0x0b,0xdb, //9
 0xe0,0x32,0x3a,0x0a,0x49,0x06,0x24,0x5c,0xc2,0xd3,0xac,0x62,0x91,0x95,0xe4,0x79, //A
 0xe7,0xc8,0x37,0x6d,0x8d,0xd5,0x4e,0xa9,0x6c,0x56,0xf4,0xea,0x65,0x7a,0xae,0x08, //B
 0xba,0x78,0x25,0x2e,0x1c,0xa6,0xb4,0xc6,0xe8,0xdd,0x74,0x1f,0x4b,0xbd,0x8b,0x8a, //C
 0x70,0x3e,0xb5,0x66,0x48,0x03,0xf6,0x0e,0x61,0x35,0x57,0xb9,0x86,0xc1,0x1d,0x9e, //D
 0xe1,0xf8,0x98,0x11,0x69,0xd9,0x8e,0x94,0x9b,0x1e,0x87,0xe9,0xce,0x55,0x28,0xdf, //E
 0x8c,0xa1,0x89,0x0d,0xbf,0xe6,0x42,0x68,0x41,0x99,0x2d,0x0f,0xb0,0x54,0xbb,0x16};//F
 return sbox[num];
}

//el conjunto de valores constantes Rcon[i], contiene los valores dados
//por x a la potencia (i-1) comenzando con la potencia de x (x es denotado
//como {02}) en el campo GF(2^8)
//nótese que se comienza de 1 y no 0).
int Rcon[255] = {
 0x8d,0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80,0x1b,0x36,0x6c,0xd8,0xab,0x4d,0x9a,
 0x2f,0x5e,0xbc,0x63,0xc6,0x97,0x35,0x6a,0xd4,0xb3,0x7d,0xfa,0xef,0xc5,0x91,0x39,
 0x72,0xe4,0xd3,0xbd,0x61,0xc2,0x9f,0x25,0x4a,0x94,0x33,0x66,0xcc,0x83,0x1d,0x3a,
 0x74,0xe8,0xcb,0x8d,0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80,0x1b,0x36,0x6c,0xd8,
 0xab,0x4d,0x9a,0x2f,0x5e,0xbc,0x63,0xc6,0x97,0x35,0x6a,0xd4,0xb3,0x7d,0xfa,0xef,
 0xc5,0x91,0x39,0x72,0xe4,0xd3,0xbd,0x61,0xc2,0x9f,0x25,0x4a,0x94,0x33,0x66,0xcc,
 0x83,0x1d,0x3a,0x74,0xe8,0xcb,0x8d,0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80,0x1b,
 0x36,0x6c,0xd8,0xab,0x4d,0x9a,0x2f,0x5e,0xbc,0x63,0xc6,0x97,0x35,0x6a,0xd4,0xb3,
 0x7d,0xfa,0xef,0xc5,0x91,0x39,0x72,0xe4,0xd3,0xbd,0x61,0xc2,0x9f,0x25,0x4a,0x94,
 0x33,0x66,0xcc,0x83,0x1d,0x3a,0x74,0xe8,0xcb,0x8d,0x01,0x02,0x04,0x08,0x10,0x20,
 0x40,0x80,0x1b,0x36,0x6c,0xd8,0xab,0x4d,0x9a,0x2f,0x5e,0xbc,0x63,0xc6,0x97,0x35,
 0x6a,0xd4,0xb3,0x7d,0xfa,0xef,0xc5,0x91,0x39,0x72,0xe4,0xd3,0xbd,0x61,0xc2,0x9f,
 0x25,0x4a,0x94,0x33,0x66,0xcc,0x83,0x1d,0x3a,0x74,0xe8,0xcb,0x8d,0x01,0x02,0x04,
 0x08,0x10,0x20,0x40,0x80,0x1b,0x36,0x6c,0xd8,0xab,0x4d,0x9a,0x2f,0x5e,0xbc,0x63,
 0xc6,0x97,0x35,0x6a,0xd4,0xb3,0x7d,0xfa,0xef,0xc5,0x91,0x39,0x72,0xe4,0xd3,0xbd,
 0x61,0xc2,0x9f,0x25,0x4a,0x94,0x33,0x66,0xcc,0x83,0x1d,0x3a,0x74,0xe8,0xcb};

// Esta función produce Nb(Nr+1) llaves de ronda. Las llaves de ronda
son usadas en cada ronda para encriptar el estado.
void KeyExpansion(){

 int i,j;
 unsigned char temp[4],k;


 // la primera clave de ronda es ella misma.
```

```
 for(i=0;i<Nk;i++){
  RoundKey[i*4]=Key[i*4];
  RoundKey[i*4+1]=Key[i*4+1];
  RoundKey[i*4+2]=Key[i*4+2];
  RoundKey[i*4+3]=Key[i*4+3];
 }
 //todas las demás claves de ronda son encontradas de la clave de ronda
 //anterior.
 while (i < (Nb * (Nr+1))){
  for(j=0;j<4;j++){
   temp[j]=RoundKey[(i-1) * 4 + j];
  }
  if (i % Nk == 0){
 // esta función rota 4 bytes de una palabra a la izquierda.
 // convierte [a0,a1,a2,a3] a [a1,a2,a3,a0]
 // Function RotWord()
    {
     k = temp[0];
     temp[0] = temp[1];
     temp[1] = temp[2];
     temp[2] = temp[3];
     temp[3] = k;
    }
 // SubWord() Toma una palabra de entrada de cuatro bytes y aplica el
 // S-box a cada uno de los cuatro bytes.
 // Function Subword()
    {
     temp[0]=getSBoxValue(temp[0]);
     temp[1]=getSBoxValue(temp[1]);
     temp[2]=getSBoxValue(temp[2]);
     temp[3]=getSBoxValue(temp[3]);
    }
    temp[0] =  temp[0] ^ Rcon[i/Nk];
  }

  else if (Nk > 6 && i % Nk == 4){
  // Function Subword()
    {
     temp[0]=getSBoxValue(temp[0]);
     temp[1]=getSBoxValue(temp[1]);
     temp[2]=getSBoxValue(temp[2]);
     temp[3]=getSBoxValue(temp[3]);
    }
  }
  RoundKey[i*4+0] = RoundKey[(i-Nk)*4+0] ^ temp[0];
  RoundKey[i*4+1] = RoundKey[(i-Nk)*4+1] ^ temp[1];
  RoundKey[i*4+2] = RoundKey[(i-Nk)*4+2] ^ temp[2];
  RoundKey[i*4+3] = RoundKey[(i-Nk)*4+3] ^ temp[3];
//se opera y guarda las letras de cada clave de ronda en la variable
//ShiftMoves
   ShiftMoves[i-Nk]=(RoundKey[i*4+0] + RoundKey[i*4+1] + RoundKey[i*4+2]
+ RoundKey[i*4+3])%2;
   i++;
 }
}

// Esta función agrega cada llave de ronda al estado.
// la llave de donda es agregada al estado con una función XOR.
void AddRoundKey(int round){
 int i,j;
 for(i=0;i<4;i++){
```

```
   for(j=0;j<4;j++){
    state[j][i] ^= RoundKey[round * Nb * 4 + i * Nb + j];
   }
  }
}

// La función SubBytes sustituye los valores de la matris de estado con
// valores de una tabla S-Box.
void SubBytes(){
 int i,j;
 for(i=0;i<4;i++){
  for(j=0;j<4;j++){
   state[i][j] = getSBoxValue(state[i][j]);
  }
 }
}

// La función ShiftRows mueve las filas de estado a la izquierda.
// para cada fila existe un diferente desplazamiento
// desplazamiento = munero de fila.
void ShiftRows(){
 unsigned char temp;
 // La primera fila es movida 1 columna a la derecha o izquierda
 // Dependiendo del valor en la variable ShiftMoves
 if(ShiftMoves[(4*round)-3]==0){
  temp=state[1][0];
  state[1][0]=state[1][1];
  state[1][1]=state[1][2];
  state[1][2]=state[1][3];
  state[1][3]=temp;
 }
 else if(ShiftMoves[(4*round)-3]==1){
  temp=state[1][3];
  state[1][3]=state[1][2];
  state[1][2]=state[1][1];
  state[1][1]=state[1][0];
  state[1][0]=temp;
 }
 // La segunda fila es movida 2 columnas a la derecha
 temp=state[2][0];
 state[2][0]=state[2][2];
 state[2][2]=temp;
 temp=state[2][1];
 state[2][1]=state[2][3];
 state[2][3]=temp;

 // La tercera fila es movida 3 columnas a la derecha o izquierda
 // Dependiendo del valor en la variable ShiftMoves
 if(ShiftMoves[(4*round)-1]==0){
  temp=state[3][0];
  state[3][0]=state[3][3];
  state[3][3]=state[3][2];
  state[3][2]=state[3][1];
  state[3][1]=temp;
 }
 else if(ShiftMoves[(4*round)-1]==1){
  temp=state[3][0];
  state[3][0]=state[3][1];
  state[3][1]=state[3][2];
  state[3][2]=state[3][3];
  state[3][3]=temp;
```

```
  }
 }

 // xtime es un macro que encuentra el producto de {02} y el argumento
 // xtime modulo {1b}
 #define xtime(x)   ((x<<1) ^ (((x>>7) & 1) * 0x1b))

 // La función MixColumns mezcla las columnas de la matriz de estado
 void MixColumns(){
  int i;
  unsigned char Tmp,Tm,t;
  for(i=0;i<4;i++){
   t=state[0][i];
   Tmp = state[0][i] ^ state[1][i] ^ state[2][i] ^ state[3][i];
   Tm = state[0][i] ^ state[1][i];
   Tm = xtime(Tm); state[0][i] ^= Tm ^ Tmp;
   Tm = state[1][i] ^ state[2][i];
   Tm = xtime(Tm); state[1][i] ^= Tm ^ Tmp;
   Tm = state[2][i] ^ state[3][i];
   Tm = xtime(Tm); state[2][i] ^= Tm ^ Tmp;
   Tm = state[3][i] ^ t;
   Tm = xtime(Tm);
   state[3][i] ^= Tm ^ Tmp;
  }
 }

 // Cipher es la función principal que encripta el texto plano.
 void Cipher(){
  int i,j,round=0;
  //copiar el texto plano al array de estado.
  for(i=0;i<4;i++){
   for(j=0;j<4;j++){
    state[j][i] = in[i*4 + j];
   }
  }

  // Agregar la primera clave de rondas al estado antes de iniciar las
  //  rondas.
  AddRoundKey(0);

  // Habrá Nr rondas.
  // Las primeras Nr-1 rondas son identicas.
  // Estas Nr-1 rondas se ejecutan en el bucle a continuacion.
  for(round=1;round<Nr;round++){
   SubBytes();
   ShiftRows();
   MixColumns();
   AddRoundKey(round);
  }

  // La última ronda se da a continuacion.
  // la función MixColumns no se ejecuta en la última ronda.
  SubBytes();
  ShiftRows();
  AddRoundKey(Nr);

  // El proceso de encriptación ha terminado.
  // Copiamos la matriz de estado a un array de salida.
  for(i=0;i<4;i++){
   for(j=0;j<4;j++){
    out[i*4+j]=state[j][i];
```

```
   }
  }
}
int main(int argc, char * argv[]){
 int i;

 // recibimos el tamaño de la clave.
 while(Nr!=128 && Nr!=192 && Nr!=256){
//  Para propósitos demostrativos el tamaño de la clave es de 128 bits.
//  Para ingresar un tamaño de clave descomentar la función scanf.
//  scanf("%d",&Nr);
   Nr=128;
 }

 // Calculamos Nk y Nr del valor recibido.
 Nk = Nr / 32;
 Nr = Nk + 6;
//leemos la clave y el archivo a ser encriptado.
 memcpy(Key, argv[1], sizeof Key);
//memcpy(in, argv[2], 16);
    FILE *input=fopen(argv[2],"r");
//el archivo encriptado saldrá con el nombre de output.txt
    outputm=fopen("output.txt","wb");

    KeyExpansion();
    int q=0;
    while(fread(in,sizeof(char),16,input)== 16){
        Cipher();
    }
 return 0;
}
```